

# CHƯƠNG 1

## Tổng quan về lập trình truyền thông

### Mục đích

Chương này nhằm cung cấp cho các bạn một cái nhìn tổng quan về các vấn đề có liên quan trong lập trình truyền thông

### Yêu cầu

Sau khi hoàn tất chương này, bạn có thể:

- Giải thích được Cơ chế giao tiếp liên quá trình (Inter-Process Communication ) là gì.
- Mô tả chức năng, nhiệm vụ của các tầng trong mô hình OSI.
- Định nghĩa về giao thức và biện luận được sự cần thiết của giao thức trong truyền thông .
- Mô tả về bộ giao thức TCP/IP.
- Định nghĩa mô hình Client – Server.
- Phân biệt được 2 chế độ giao tiếp: Nghẽn và Không nghẽn.
- Phân biệt được các kiểu kiến trúc chương trình.

## 1.1. Cơ chế giao tiếp liên quá trình là gì ?

Truyền thông là một khái niệm dùng để chỉ sự giao tiếp, trao đổi thông tin giữa hai hay nhiều thực thể trong một hệ thống nào đó. Nếu hệ thống mà chúng ta xem xét là xã hội loài người, thì truyền thông có thể là quá trình trao đổi thông tin giữa người với người trong cuộc sống thông qua các phương tiện truyền tải thông tin khác nhau như không khí (trong trò chuyện trực tiếp), hệ thống điện thoại, sách, báo, các phương tiện nghe nhìn, mạng máy tính...

Nếu hệ thống mà chúng ta xem xét là một hệ thống máy tính hay một hệ thống mạng thì truyền thông có thể được phân thành hai mức:

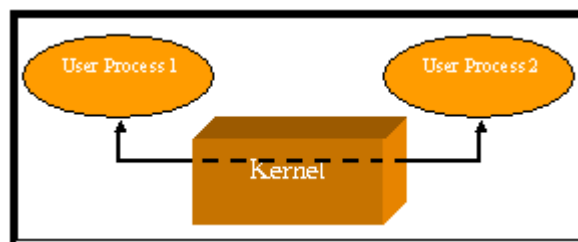
- Mức phần cứng: là sự giao tiếp, trao đổi thông tin giữa các bộ phận vật lý cấu thành nên hệ thống máy tính như CPU, bộ nhớ, thiết bị vào ra, card giao tiếp mạng, nhờ vào các phương tiện truyền thông như hệ thống BUS nội, hệ thống BUS vào ra hay các dây cáp mạng . . .
- Mức phần mềm: là sự giao tiếp, trao đổi thông tin giữa các thành phần bên trong của một chương trình hay giữa các chương trình với nhau thông qua các cơ chế truyền thông được hỗ trợ bởi các hệ điều hành, hệ điều hành mạng.

Trong các hệ thống máy tính đơn nhiệm (monotasking) cổ điển, ví dụ MS-DOS, tại một thời điểm chỉ cho phép tồn tại một quá trình. Việc giao tiếp, trao đổi thông tin chỉ diễn ra trong phạm vi của một chương trình. Đó là sự giao tiếp giữa các thủ tục dưới hình thức chia sẻ các biến toàn cục, hay bằng cách truyền các tham số khi gọi hàm, thủ tục hay bằng giá trị trả về của một hàm . . . Ngược lại, trong các hệ thống đa nhiệm (multitasking) có nhiều quá trình tồn tại song song nhau, mỗi quá trình được thực hiện trong một không gian địa chỉ (Address space) riêng biệt. Việc giao tiếp giữa các quá trình muốn thực hiện được đòi hỏi phải có những tiện ích hỗ trợ bởi hệ điều hành, hệ điều hành mạng. Các tiện ích này thường được gọi với cái tên là **Cơ chế giao tiếp liên quá trình** (IPC - Inter-Process Communication).

## 1.2. Phân loại cơ chế giao tiếp liên quá trình

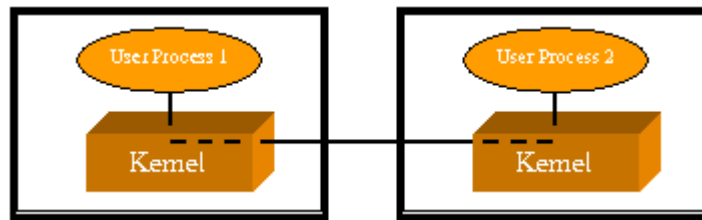
Các cơ chế giao tiếp liên quá trình được hỗ trợ bởi các hệ điều hành đa nhiệm, hệ điều hành mạng được chia ra làm hai loại:

- Loại 1: Cơ chế giao tiếp liên quá trình hỗ trợ giao tiếp giữa các quá trình trên cùng một máy tính. (Hình H1.1)



Hình 1.1 - Cơ chế giao tiếp liên quá trình loại 1

- Loại 2: Cơ chế giao tiếp liên quá trình hỗ trợ giao tiếp giữa các quá trình nằm trên các máy tính khác nhau (Hình H1.2).



Hình 1.2 - Cơ chế giao tiếp liên quá trình loại 2

Trong cơ chế giao tiếp liên quá trình trên cùng một máy, dữ liệu trao đổi qua lại giữa các quá trình phải đi xuyên qua hạt nhân (kernel) của hệ điều hành. Đó có thể là một vùng nhớ dùng chung cho các quá trình đã được qui định trước bởi hệ điều hành, hay một tập tin trên đĩa được quản lý bởi hệ điều hành trong đó một quá trình sẽ ghi dữ liệu vào, quá trình khác đọc dữ liệu ra, . . .

Trong cơ chế giao tiếp liên quá trình trên các máy tính khác nhau, dữ liệu trao đổi giữa các quá trình không những phải đi qua hạt nhân như cơ chế giao tiếp liên quá trình trên một máy tính mà hơn thế các hạt nhân của các máy có liên quan phải hiểu nhau. Nói cách khác các hạt nhân phải thoả thuận trước với nhau về các qui tắc trao đổi thông tin giữa chúng. Thông thường ta gọi các qui tắc này là các [giao thức](#) (Protocol).

### 1.3. Mô hình tham khảo OSI

Để dễ dàng cho việc nối kết và trao đổi thông tin giữa các máy tính với nhau, vào năm 1983, Tổ chức tiêu chuẩn thế giới ISO đã phát triển một mô hình cho phép hai máy tính có thể gửi và nhận dữ liệu cho nhau. Mô hình này dựa trên tiếp cận phân tầng (lớp), với mỗi tầng đảm nhiệm một số các chức năng cơ bản nào đó và được gọi là mô hình OSI.

Để hai máy tính có thể trao đổi thông tin được với nhau cần có rất nhiều vấn đề liên quan. Ví dụ như cần có Card mạng, dây cáp mạng, điện thế tín hiệu trên cáp mạng, cách thức đóng gói dữ liệu, điều khiển lỗi đường truyền ... Bằng cách phân chia các chức năng này vào những tầng riêng biệt nhau, việc viết các phần mềm để thực hiện chúng trở nên dễ dàng hơn. Mô hình OSI giúp đồng nhất các hệ thống máy tính khác biệt nhau khi chúng trao đổi thông tin. Mô hình này gồm có 7 tầng:

#### 7. Tầng ứng dụng (Application Layer)

Đây là tầng trên cùng, cung cấp các ứng dụng truy xuất đến các dịch vụ mạng. Nó bao gồm các ứng dụng của người dùng, ví dụ như các Web Browser (Netscape Navigator, Internet Explorer), các Mail User Agent (Outlook Express, Netscape Messenger, ...) hay các chương trình làm server cung cấp các dịch vụ mạng như các Web Server (Netscape Enterprise, Internet Information Service, Apache, ...), Các FTP Server, các Mail server (Send mail, MDeamon). Người dùng mạng giao tiếp trực tiếp với tầng này.

#### 6. Tầng trình bày (Presentation Layer)

Tầng này đảm bảo các máy tính có kiểu định dạng dữ liệu khác nhau vẫn có thể trao đổi thông tin cho nhau. Thông thường các máy tính sẽ thống nhất với nhau về một kiểu định dạng dữ liệu trung gian để trao đổi thông tin giữa các máy tính. Một dữ liệu cần gửi đi sẽ được tầng trình bày chuyển sang định dạng trung gian trước khi nó được truyền lên mạng. Ngược lại, khi nhận dữ liệu từ mạng, tầng trình bày sẽ chuyển dữ liệu sang định dạng riêng của nó.

#### 5. Tầng giao dịch (Session Layer)

Tầng này cho phép các ứng dụng thiết lập, sử dụng và xóa các kênh giao tiếp giữa chúng (được gọi là giao dịch). Nó cung cấp cơ chế cho việc nhận biết tên và các chức năng về bảo mật thông tin khi truyền qua mạng.

#### 4. Tầng vận chuyển (Transport Layer)

Tầng này đảm bảo truyền tải dữ liệu giữa các quá trình. Dữ liệu gửi đi được đảm bảo không có lỗi, theo đúng trình tự, không bị mất mát, trùng lặp. Đối với các gói tin có kích thước lớn, tầng này sẽ phân chia chúng thành các phần nhỏ trước khi gửi đi, cũng như tập hợp lại chúng khi nhận được.

#### 3. Tầng mạng (Network Layer)

Tầng này đảm bảo các gói dữ liệu (Packet) có thể truyền từ máy tính này đến máy tính kia cho dù không có đường truyền vật lý trực tiếp giữa chúng. Nó nhận nhiệm vụ tìm đường đi cho dữ liệu đến các đích khác nhau trong hệ thống mạng.

#### 2. Tầng liên kết dữ liệu (Data-Link Layer)

Tầng này đảm bảo truyền tải các khung dữ liệu (Frame) giữa hai máy tính có đường truyền vật lý nối trực tiếp với nhau. Nó cài đặt cơ chế phát hiện và xử lý lỗi dữ liệu nhận.

#### 1. Tầng vật lý (Physical Layer)

Điều khiển việc truyền tải thật sự các bit trên đường truyền vật lý. Nó định nghĩa các tín hiệu điện, trạng thái đường truyền, phương pháp mã hóa dữ liệu, các loại đầu nối được sử dụng.

Về nguyên tắc, tầng n của một hệ thống chỉ giao tiếp, trao đổi thông tin với tầng n của hệ thống khác. Mỗi tầng sẽ có các đơn vị truyền dữ liệu riêng:

- Tầng vật lý: bit
- Tầng liên kết dữ liệu: Frame
- Tầng mạng: Packet
- Tầng vận chuyển: Segment

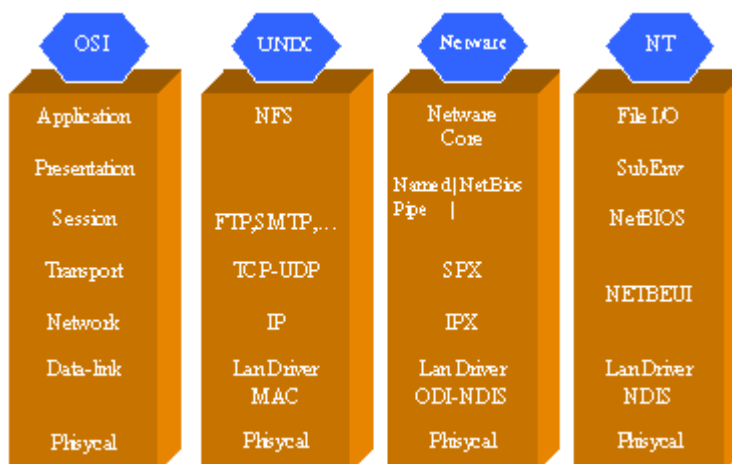
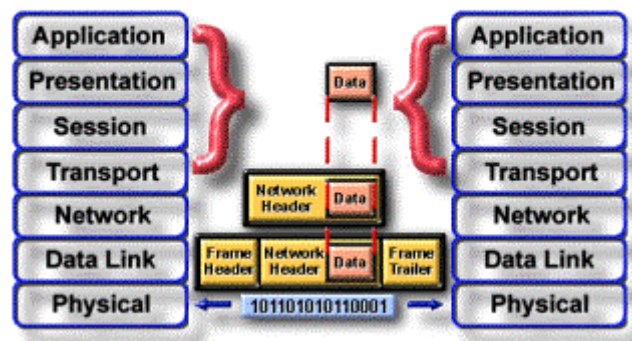
Trong thực tế, dữ liệu được gửi đi từ tầng trên xuống tầng dưới cho đến tầng thấp nhất của máy tính gửi. Ở đó, dữ liệu sẽ được truyền đi trên đường truyền vật lý. Mỗi khi

dữ liệu được truyền xuống tầng phía dưới thì nó bị "gói" lại trong đơn vị dữ liệu của tầng dưới. Tại bên nhận, dữ liệu sẽ được truyền ngược lên các tầng cao dần. Mỗi lần qua một tầng, đơn vị dữ liệu tương ứng sẽ được "tháo" ra.

Đơn vị dữ liệu của mỗi tầng sẽ có một tiêu đề (header) riêng, được mô tả trong hình 1.3.

OSI chỉ là mô hình tham khảo, mỗi nhà sản xuất khi phát minh ra hệ thống mạng của mình sẽ thực hiện các chức năng ở từng tầng theo những cách thức riêng. Các cách thức này thường được mô tả dưới dạng các chuẩn mạng hay các giao thức mạng. Như vậy dẫn đến trường hợp cùng một chức năng nhưng hai hệ thống mạng khác nhau sẽ không tương tác được với nhau. Hình 1.4 sẽ so sánh kiến trúc của các hệ điều hành mạng thông dụng với mô hình OSI.

Hình 1.3 - Xử lý dữ liệu qua các tầng



Hình 1.4 - Kiến trúc của một số hệ điều hành mạng thông dụng

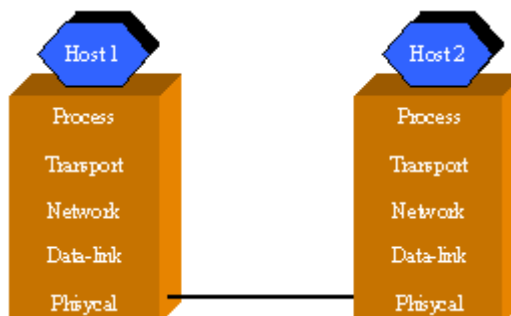
Để thực hiện các chức năng ở tầng 3 và tầng 4 trong mô hình OSI, mỗi hệ thống mạng sẽ có các protocol riêng:

- UNIX: Tầng 3 dùng giao thức IP, tầng 4 giao thức TCP/UDP
- Netware: Tầng 3 dùng giao thức IPX, tầng 4 giao thức SPX
- Windows NT: chỉ dùng 1 giao thức NETBEUI

Nếu chỉ dừng lại ở đây thì các máy tính UNIX, Netware, NT sẽ không trao đổi thông tin được với nhau. Với sự lớn mạnh của mạng Internet, các máy tính cài đặt các hệ điều hành khác nhau đòi hỏi phải giao tiếp được với nhau, tức phải sử dụng chung một giao thức. Đó chính là bộ giao thức TCP/IP, giao thức của mạng Internet.

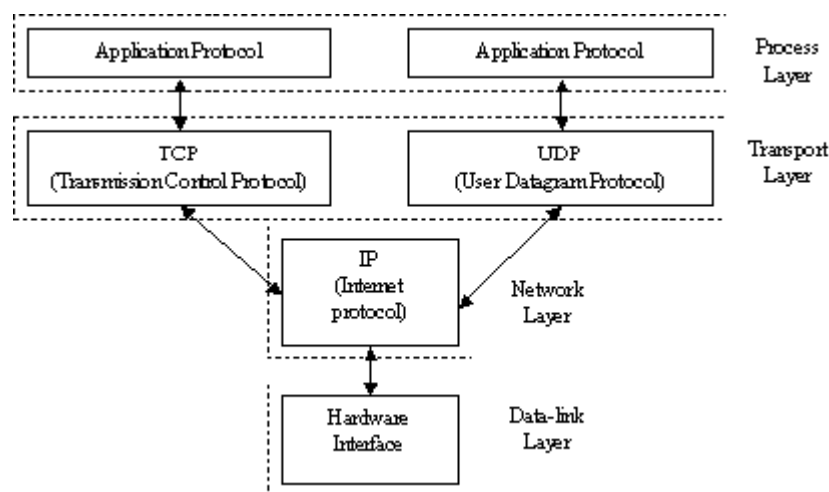
## 1.4. Mạng TCP/IP

Đây là kiến trúc của mạng Internet, chỉ gồm 5 tầng như hình vẽ dưới đây:



Hình 1.5 - Kiến trúc mạng TCP/IP

Người ta còn gọi mô hình này là mô hình OSI đơn giản. Các giao thức được sử dụng trên mỗi tầng được qui định như sau:



Hình 1.6 – Bộ giao thức TCP/IP

Tầng 3 sử dụng giao thức IP, tầng 4 có thể sử dụng giao thức TCP ở chế độ có nối kết hoặc UDP ở chế độ không nối kết.

Tầng 5 là tầng của các ứng dụng. Mỗi loại ứng dụng phải định nghĩa một giao thức riêng để các thành phần trong ứng dụng trao đổi thông tin qua lại với nhau. Một số ứng dụng phổ biến đã trở thành chuẩn của mạng Internet như:

- Ứng dụng Web: Sử dụng giao thức HTTP để tải các trang web từ Web Server về Web Browser.
- Ứng dụng thư điện tử: Sử dụng giao thức SMTP để chuyển tiếp mail gửi đi đến Mail Server của người nhận và dùng giao thức POP3 hoặc IMAP để nhận mail về cho người đọc.
- Ứng dụng truyền tải tập tin: Sử dụng giao thức FTP để tải (download) các tập tin từ các FTP Server ở xa về máy người dùng hay ngược lại.
- .....

Thông thường các tầng 1,2,3 và 4 được phát triển bởi các nhà sản xuất hệ điều hành, nhà sản xuất các thiết bị phần cứng mạng. Chúng đảm nhận nhiệm vụ truyền tải thông tin cho các quá trình trên tầng ứng dụng. Chúng cài đặt các cơ chế giao tiếp liên quá trình để các quá trình trên tầng ứng dụng có thể truy xuất đến dịch vụ truyền tải thông tin do chúng cung cấp. Trong khi đó, tầng 5 là nơi các nhà sản xuất phần mềm khai thác để tạo ra các ứng dụng giải quyết các vấn đề khác nhau của cuộc sống. Nó được xem như là tầng xử lý thông tin.

## 1.5. Dịch vụ mạng

Dịch vụ mạng (Net service) là một chương trình ứng dụng thực hiện một tác vụ nào đó trên hệ thống mạng.

### Ví dụ:

- Dịch vụ in trên mạng cho phép nhiều máy tính cùng sử dụng một máy in.
- Dịch vụ tập tin trên mạng cho phép chia sẻ chương trình, dữ liệu giữa các máy tính.
- Dịch vụ web cung cấp các trang web cho các máy tính khác nhau

Có nhiều mô hình khác nhau để xây dựng các dịch vụ mạng. Một trong những mô hình được sử dụng khá phổ biến là mô hình Client-Server. Đây là một mô hình cơ bản để xây dựng các dịch vụ mạng.

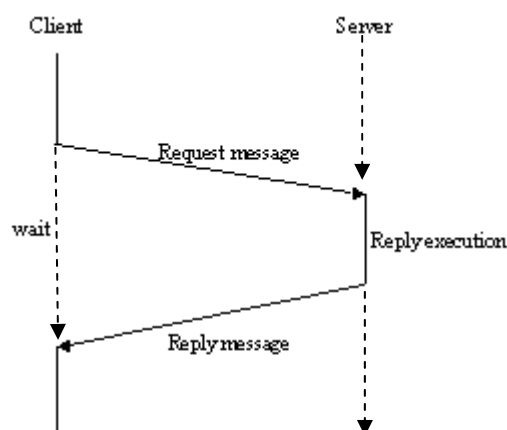
## 1.6. Mô hình Client – Server

### 1.6.1. Giới thiệu

Trong mô hình này, chương trình ứng dụng được chia thành 2 thành phần:

- Quá trình chuyên cung cấp một số phục vụ nào đó, chẳng hạn: phục vụ tập tin, phục vụ máy in, phục vụ thư điện tử, phục vụ Web... Các quá trình này được gọi là các trình phục vụ hay **Server**.
- Một số quá trình khác có yêu cầu sử dụng các dịch vụ do các server cung cấp được gọi là các quá trình khách hàng hay **Client**.

Việc giao tiếp giữa client và server được thực hiện dưới hình thức trao đổi các thông điệp (Message). Để được phục vụ, client sẽ gửi một thông điệp yêu cầu (Request Message) mô tả về công việc muốn server thực hiện. Khi nhận được thông điệp yêu cầu, server tiến hành phân tích để xác định công việc cần phải thực thi. Nếu việc thực hiện yêu cầu này có sinh ra kết quả trả về, server sẽ gửi nó cho client trong một thông điệp trả lời (Reply Message). Dạng thức (format) và ý nghĩa của các thông điệp



Hình 1.7 – Mô hình Client-Server

————— Quá trình đang được thực thi  
..... Quá trình đang bị ngừng

trao đổi giữa client và server được qui định rõ bởi giao thức (protocol) của ứng dụng.

### 1.6.2. Ví dụ về dịch vụ Web.

Dịch vụ web được tổ chức theo mô hình Client -Server, trong đó:

- Web server sẵn sàng cung cấp các trang web đang được lưu trữ trên đĩa cứng cục bộ của mình.
- Web Client, còn gọi là các Browser, có nhu cầu nhận các trang web từ các Web Server
- HTTP là giao thức trao đổi thông tin qua lại giữa Web client và Web Server.
- Thông điệp yêu cầu là một chuỗi có dạng sau:

```
Command URL HTTP/Ver \n\n
```

- Thông điệp trả lời có dạng sau:

```
<HEADER>\n\n
```

```
<CONTENT>
```

- Giả sử Client cần nhận trang Web ở địa chỉ <http://www.cit.ctu.edu.vn/>, nó sẽ gửi đến Web Server có tên [www.cit.ctu.edu.vn](http://www.cit.ctu.edu.vn/) thông điệp yêu cầu sau:

```
GET www.cit.ctu.edu.vn HTTP/1.1\n\n
```

- Server sẽ gửi về nội dung sau:

```
HTTP/1.0 200 OK
Date: Mon, 24 Nov 2003 02:43:46 GMT
Server: Apache/1.3.23 (Unix) (Red-Hat/Linux) mod_ssl/2.8.7
OpenSSL/0.9.6b DAV/1
.0.3 PHP/4.1.2 mod_perl/1.26
Last-Modified: Tue, 01 Jul 2003 08:08:52 GMT
ETag: "17f5d-2abb-3f014194"
Accept-Ranges: bytes
Content-Length: 10939
Content-Type: text/html
X-Cache: HIT from proxy.cit.ctu.edu.vn
Proxy-Connection: close
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">
```

```
<HTML>
```

```
<HEAD>
```

```
<LINK href="favicon.ico" rel="SHORTCUT ICON">
```

.....



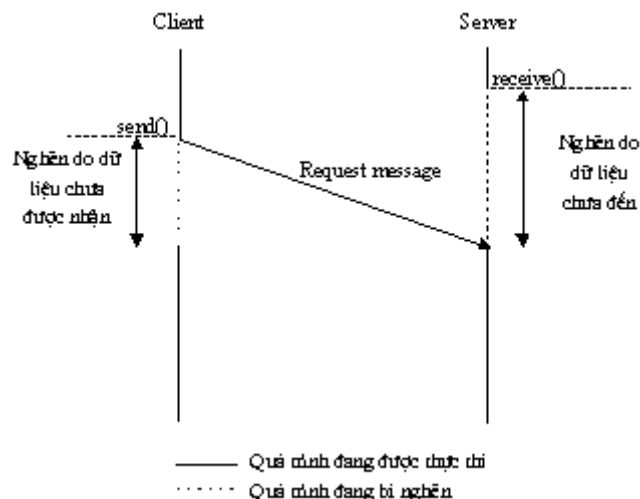
### 1.6.3. Các chế độ giao tiếp

Quá trình giao tiếp giữa client và server có thể diễn ra theo hai chế độ là nghẽn (blocked) hay không nghẽn (Non blocked).

#### 1.6.3.1. Chế độ nghẽn :

Trong chế độ này, khi quá trình client hay server phát ra lệnh gọi dữ liệu, (thông thường bằng lệnh send) , sự thực thi của nó sẽ bị tạm dừng cho đến khi quá trình nhận phát ra lệnh nhận số dữ liệu đó (thường là lệnh receive).

Tương tự cho trường hợp nhận dữ liệu, nếu quá trình nào đó, client hay server, phát ra lệnh nhận dữ liệu, mà ở thời điểm đó chưa có dữ liệu gửi đến, sự thực thi của nó cũng tạm dừng cho đến khi có dữ liệu gửi đến.



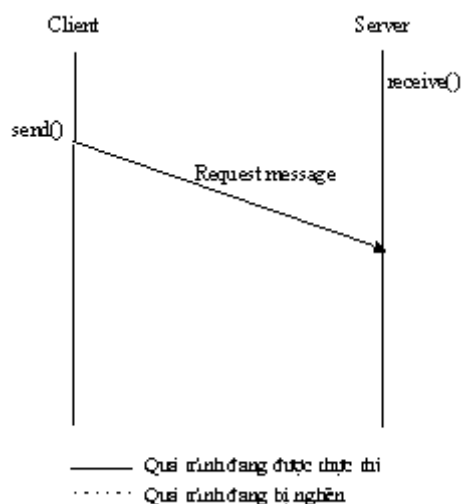
Hình 1.8 - Chế độ giao tiếp nghẽn

#### 1.6.3.2. Chế độ không nghẽn:

Trong chế độ này, khi quá trình client hay server phát ra lệnh gọi dữ liệu, sự thực thi của nó vẫn được tiếp tục mà không quan tâm đến việc có quá trình nào phát ra lệnh nhận số dữ liệu đó hay không.

Tương tự cho trường hợp nhận dữ liệu, khi quá trình phát ra lệnh nhận dữ liệu, nó sẽ nhận được số lượng dữ liệu hiện có (bằng 0 nếu chưa có quá trình nào gửi dữ liệu đến). Sự thực thi của quá trình vẫn được tiếp tục.

Trong thực tế cần chú ý đến chế độ giao tiếp nghẽn khi lập trình, vì nó có thể dẫn đến trường hợp chương trình bị "treo" do số lần gửi và nhận giữ liệu không bằng nhau giữa hai bên giao tiếp.



Hình 1.9 - Chế độ giao tiếp không nghẽn

## 1.7. Các kiểu kiến trúc chương trình

Ở mức luận lý, các chức năng mà một chương trình ứng dụng thực hiện có thể xếp thành một trong 3 loại sau:

1. Các chức năng thực hiện việc tương tác với người dùng như tạo các giao diện nhập liệu hay in các báo biểu, thông báo ra màn hình. Các chức năng này được gọi chung là **Dịch vụ giao diện người dùng** (User Interface Service).

2. Các chức năng tính toán các dữ liệu, xử lý thông tin theo những qui luật (rule), giải thuật được qui định bởi vấn đề mà ứng dụng giải quyết. Các chức năng này được gọi chung là **Dịch vụ nghiệp vụ** (Business Rule Service).
3. Trong quá trình tính toán, chương trình ứng dụng cần truy vấn đến các thông tin đã có được lưu trên đĩa cứng hay trong các cơ sở dữ liệu. Cũng như cần thiết phải lưu lại các kết quả tính toán được để sử dụng về sau. Các chức năng này được gọi chung là **Dịch vụ lưu trữ** (Data Storage Service).

Ở mức vật lý, các chức năng này có thể được cài đặt vào một hay nhiều tập tin thực thi hình thành các kiểu kiến trúc chương trình khác nhau. Cho đến thời điểm hiện nay, người ta chia kiến trúc của chương trình thành 3 loại được trình bày tiếp theo sau.

### 1.7.1. Kiến trúc đơn tầng (Single-tier Architecture)

Trong kiểu kiến trúc này, cả 3 thành phần của chương trình ứng dụng (User Interface, Business Rule, Data Storage) đều được tích hợp vào một tập tin thực thi.

Ví dụ: BKAV, D2, Winword, . . .

Các ứng dụng kiểu này chỉ được thực thi trên một máy tính.



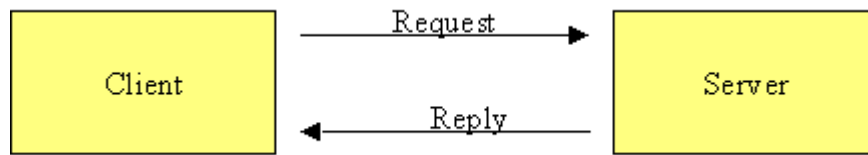
Hình 1.10 - Kiến trúc chương trình đơn tầng

- **Ưu điểm:**
  - Dễ dàng trong thiết kế cài đặt ứng dụng kiểu này.
- **Nhược điểm:**
  - Bởi vì cả 3 thành phần được cài vào một tập tin thực thi, nên việc sửa lỗi hay nâng cấp chương trình thì rất khó khăn. Toàn bộ chương trình phải biên dịch lại cho dù chỉ sửa đổi một lỗi rất nhỏ trong một thành phần nào đó ( User Interface chẳng hạn).
  - Việc bảo trì, nâng cấp ấn bản mới là một công việc cực kỳ nặng nề vì ta phải thực hiện việc cài đặt trên tất cả các máy tính.
  - Trong kiểu này, mỗi máy tính duy trì một cơ sở dữ liệu riêng cho nên rất khó trong việc trao đổi, tổng hợp dữ liệu.
  - Máy tính phải đủ mạnh để có thể thực hiện đồng thời cả 3 loại dịch vụ.

### 1.7.2. Kiến trúc hai tầng (Two - Tier Architecture)

Kiến trúc này còn được biết đến với tên kiến trúc Client-Server. Kiến trúc này gồm 2 chương trình thực thi: chương trình Client và chương trình Server. Cả hai chương trình có thể được thực thi trên cùng một máy tính hay trên hai máy tính khác nhau.

Client và Server trao đổi thông tin với nhau dưới dạng các thông điệp (Message). Thông điệp gửi từ Client sang Server gọi là các thông điệp yêu cầu (Request Message) mô tả công việc mà phần Client muốn Server thực hiện.



Hình 1.11 - Kiến trúc chương trình Client-Server

Mỗi khi Server nhận được một thông điệp yêu cầu, Server sẽ phân tích yêu cầu, thực thi công việc theo yêu cầu và gửi kết quả về client (nếu có) trong một thông điệp trả lời (Reply Message). Khi vận hành, một máy tính làm Server phục vụ cho nhiều máy tính Client.

Mỗi một ứng dụng Client-Server phải định nghĩa một **Giao thức** (Protocol) riêng cho sự trao đổi thông tin, phối hợp công việc giữa Client và Server. Protocol qui định một số vấn đề cơ bản sau:

- Khuôn dạng loại thông điệp.
- Số lượng và ý nghĩa của từng loại thông điệp.
- Cách thức bắt tay, đồng bộ hóa tiến trình truyền nhận giữa Client và Server.
- . . . .

Thông thường phần client đảm nhận các chức năng về User Interface, như tạo các form nhập liệu, các thông báo, các báo biểu giao tiếp với người dùng.

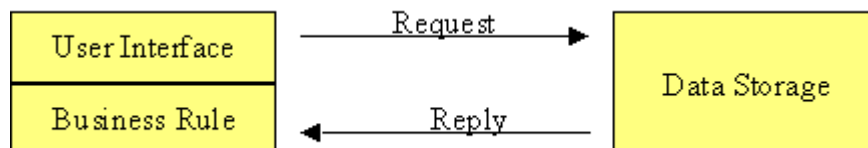
Phần Server đảm nhận các chức năng về Data Storage. Nhờ đó dễ dàng trong việc bảo trì, chia sẻ tổng hợp dữ liệu trong toàn bộ công ty hoặc tổ chức.

Các chức năng về Business Rule có thể được cài đặt ở phần client hoặc ở phần server tạo ra hai loại kiến trúc Client - Server là:

- Fat Client
- Fat Server.

#### 1.7.2.1. Loại Fat Client

Trong loại này Business Rule được cài đặt bên phía Client. Phần Server chủ yếu thực hiện chức năng về truy vấn và lưu trữ thông tin.



Hình 1.12 - Kiến trúc chương trình Client – Server theo kiểu Fat Client

Ưu điểm

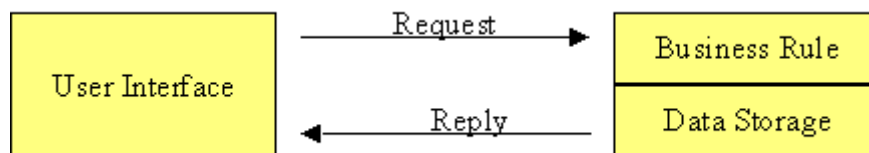
- Tạo ra ít giao thông trên mạng nhờ dữ liệu tạm thời trong quá trình tính toán được lưu tại Client.

#### Nhược điểm

- Vì Business Rule được cài đặt trên phía Client, đòi hỏi máy tính thực thi phần Client phải đủ mạnh, dẫn đến tốn kém trong chi phí đầu tư phần cứng cho các công ty xí nghiệp.
- Phải cài lại tất cả các máy tính Client khi nâng cấp chương trình.

#### 1.7.2.2. Loại Fat Server

Trong loại này, phần lớn các chức năng về Business Rule được đặt ở phần Server. Phần Client chỉ thực hiện một số chức năng nhỏ của Business Rule về kiểm tra tính hợp lệ của dữ liệu nhập bởi người dùng.



Hình 1.13 - Kiến trúc chương trình Client – Server theo kiểu Fat Server

#### Ưu điểm

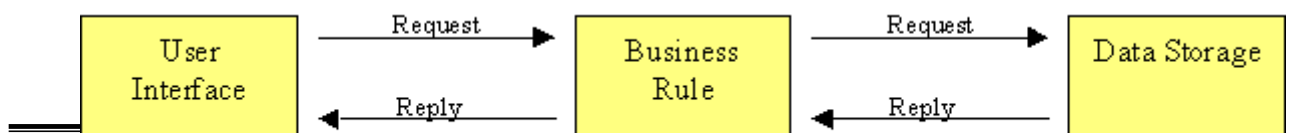
- Vì Business Rule được đặt ở phần Server, các máy tính Client không cần phải có cấu hình mạnh.
- Việc nâng cấp chương trình khi Business Rule thay đổi trở nên nhẹ nhàng hơn vì chỉ phải cài đặt lại phần Server.

#### Nhược điểm

- Tạo ra nhiều thông điệp trao đổi giữa Client và Server làm tăng giao thông trên mạng.
- Tăng tải trên máy Server vì nó phải đồng thời thực hiện các chức năng của Business Rule và Data Storage làm giảm hiệu năng của chương trình.

#### 1.7.3. Kiến trúc đa tầng (N-Tier Architecture)

Đây là kiến trúc cho các **Ứng dụng phân tán** (Distributed Application). Thông thường là kiến trúc 3 tầng. Chương trình ứng dụng được tách thành 3 phần riêng biệt tương ứng cho 3 chức năng User Interface, Business Rule và Data Storage. Vì các chức năng thuộc về Business Rule được tách thành một phần riêng, nó có thể được thực thi trên một máy tính Server riêng giải quyết được hầu hết các nhược điểm mắc phải của kiến trúc đơn tầng và kiến trúc hai tầng nói trên.



### Hình 1.12 - Kiến trúc chương trình đa tầng

Kiến trúc này đáp ứng tốt với những thay đổi về qui luật xử lý dữ liệu của vấn đề mà ứng dụng giải quyết. Việc thay đổi chỉ ảnh hưởng trên tầng Business Rule mà không ảnh hưởng đến hai tầng còn lại.

Thông thường, người ta gọi tên các thành phần trên là:

Client – Application Server – Database Server

## 1.8. Bài tập

### 1.8.1. Bài tập bắt buộc

#### **Bài tập 1.1: Protocol HTTP**

Tìm đọc và viết một báo cáo không quá 10 trang về giao thức HTTP.

#### **Bài tập 1.2: Chat Protocol**

Tìm hiểu về dịch vụ Chat trên mạng Internet. Viết một bảng báo cáo không qua 10 trang trình bày 2 nội dung sau:

- Một bảng mô tả các chức năng thường được hỗ trợ trong một dịch vụ Chat.
- Xây dựng Chat Protocol riêng của bạn trong đó mô tả:
  - Các chức năng hỗ trợ bởi Chat Server.
  - Khuôn dạng (Format) và các loại thông điệp (Message) hỗ trợ bởi Protocol.
  - Sơ đồ trạng thái hoạt động của server và client (giải thuật).
  - Minh họa các kịch bản khác nhau cho từng chức năng của dịch vụ.

### 1.8.2. Bài tập gợi ý

#### **Bài tập 1.3: POP3 Protocol**

Tìm đọc và viết một báo cáo không quá 10 trang về giao thức POP3.

# CHƯƠNG 2

## Sơ lược về ngôn ngữ Java

### Mục đích

Chương này nhằm giới thiệu sơ lược về ngôn ngữ java cho các sinh viên đã có kiến thức căn bản về Lập trình hướng đối tượng với C++. Chương này sẽ không đề cập đến tất cả các vấn đề có trong Java mà chỉ giới thiệu những vấn đề cơ bản nhất về ngôn ngữ Java, đủ để các học viên có thể đọc hiểu các chương trình minh họa và làm được các bài tập ứng dụng ở các chương sau.

### Yêu cầu

Sau khi hoàn tất chương này, bạn có thể:

- Trình bày được những vấn đề tổng quan về ngôn ngữ Java như:
  - Đặc điểm và khả năng của ngôn ngữ Java.
  - Khái niệm máy ảo của Java (JVM - Java Virtual Machine ).
  - Vai trò của bộ phát triển ứng dụng JDK (Java Development Kit).
  - Phân biệt được hai kiểu chương trình Applet và Application của Java.
  - Các kiểu dữ liệu và các phép toán được hỗ trợ bởi Java.
- Biên soạn, biên dịch và thực thi thành công chương trình HelloWorld .
- Sử dụng thành thạo các cấu trúc điều khiển dưới Java như: if, switch, while, do-while, for.
- Biết cách nhận đối số của chương trình Java.
- Biết đổi chuỗi thành số trong Java.
- Sử dụng được cơ chế ngoại lệ của Java.
- Biết định nghĩa lớp mới, sử dụng một lớp đã có của Java.
- Giải thích được cơ chế vào ra với Stream trong Java.
- Sử dụng thành thạo các phương thức của hai lớp InputStream và OutputStream.
- Có thể nhập / xuất chuỗi trên một InputStream / OutputStream.
- Giải thích được cơ chế luồng (Thread).
- Cài đặt được các luồng trong Java.

## 1.1. Giới thiệu về ngôn ngữ Java

### 1.1.1. Lịch sử phát triển

Năm 1990, Sun Microsystems thực hiện dự án Green nhằm phát triển phần mềm trong các thiết bị dân dụng. James Gosling, chuyên gia lập trình đã tạo ra một ngôn ngữ lập trình mới có tên là Oak. Ngôn ngữ này có cú pháp gần giống như C++ nhưng bỏ qua các tính năng nguy hiểm của C++ như truy cập trực tiếp tài nguyên hệ thống, con trỏ, định nghĩa chồng các tác tử...

Khi ngôn ngữ Oak trưởng thành, WWW cũng đang vào thời kỳ phát triển mạnh mẽ, Sun cho rằng đây là một ngôn ngữ thích hợp cho Internet. Năm 1995, Oak đổi tên thành Java và sau đó đến 1996 Java đã được xem như một chuẩn công nghiệp cho Internet.

### 1.1.2. Khả năng của ngôn ngữ Java

- Là một ngôn ngữ bậc cao như C, C++, Perl, SmallTalk,... cho nên có thể được dùng để tạo ra các ứng dụng để giải quyết các vấn đề về số, xử lý văn bản, tạo ra trò chơi, và nhiều thứ khác.
- Có các môi trường lập trình đồ họa như Visual Java, Symantec Cafe, Jbuilder, Jcreator, ...
- Có khả năng truy cập dữ liệu từ xa thông qua cầu nối JDBC (Java DataBase Connectivity)
- Hỗ trợ các lớp hữu ích, tiện lợi trong lập trình các ứng dụng mạng (Socket) cũng như truy xuất Web.
- Hỗ trợ lập trình phân tán (Remote Method Invocation) cho phép một ứng dụng có thể được xử lý phân tán trên các máy tính khác nhau.
- Và luôn được bổ sung các tính năng cao cấp khác trong các phiên bản sau.

### 1.1.2. Những đặc điểm của ngôn ngữ Java

- Ngôn ngữ hoàn toàn hướng đối tượng.
- Ngôn ngữ đa nền cho phép một chương trình có thể thực thi trên các hệ điều hành khác nhau (MS Windows, UNIX, Linux) mà không phải biên dịch lại chương trình. Phương châm của java là **"Viết một lần, Chạy trên nhiều nền"** (Write Once, Run Anywhere).
- Ngôn ngữ đa luồng, cho phép trong một chương trình có thể có nhiều luồng điều khiển được thực thi song song nhau, rất hữu ích cho các xử lý song song.
- Ngôn ngữ phân tán, cho phép các đối tượng của một ứng dụng được phân bố và thực thi trên các máy tính khác nhau.
- Ngôn ngữ động, cho phép mã lệnh của một chương trình được tải từ một máy tính về máy của người yêu cầu thực thi chương trình.
- Ngôn ngữ an toàn, tất cả các thao tác truy xuất vào các thiết bị vào ra đều thực hiện trên máy ảo nhờ đó hạn chế các thao tác nguy hiểm cho máy tính thật.
- Ngôn ngữ đơn giản, dễ học, kiến trúc chương trình đơn giản, trong sáng.

### 1.1.3. Máy ảo Java (JMV - Java Virtual Machine)

Để đảm bảo tính đa nền, Java sử dụng cơ chế **Máy ảo của Java**. ByteCode đó là ngôn ngữ máy của Máy ảo Java tương tự như các lệnh nhị phân của các máy tính thực.

Một chương trình sau khi được viết bằng ngôn ngữ Java (có phần mở rộng là .java) phải được biên dịch thành tập tin thực thi được trên máy ảo Java (có phần mở rộng là .class). Tập tin thực thi này chứa các chỉ thị dưới dạng mã Bytecode mà máy ảo Java hiểu được phải làm gì.

Khi thực hiện một chương trình, máy ảo Java lần lượt thông dịch các chỉ thị dưới dạng Bytecode thành các chỉ thị dạng nhị phân của máy tính thực và thực thi thực sự chúng trên máy tính thực.

Máy ảo thực tế đó là một chương trình thông dịch. Vì thế các hệ điều hành khác nhau sẽ có các máy ảo khác nhau. Để thực thi một ứng dụng của Java trên một hệ điều hành cụ thể, cần phải cài đặt máy ảo tương ứng cho hệ điều hành đó.

#### 1.1.4. Hai kiểu ứng dụng dưới ngôn ngữ java

Khi bắt đầu thiết kế một ứng dụng dưới ngôn ngữ Java, bạn phải chọn kiểu cho nó là **Application** hay **Applet**.

- **Applet:** Là một chương trình ứng dụng được nhúng vào các trang web. Mã của chương trình được tải về máy người dùng từ Web server khi người dùng truy xuất đến trang web chứa nó.
- **Application:** Là một chương trình ứng dụng được thực thi trực tiếp trên các máy ảo của Java.

#### 1.1.5. Bộ phát triển ứng dụng Java (JDK- Java Development Kit)

JDK là một bộ công cụ cho phép người lập trình phát triển và triển khai các ứng dụng bằng ngôn ngữ java được cung cấp miễn phí bởi công ty JavaSoft (hoặc Sun). Có các bộ Jdk cho các hệ điều hành khác nhau. Các ấn bản của JDK không ngừng được phát hành, các bạn có thể tải về từ địa chỉ <http://java.sun.com> hoặc <http://www.javasoft.com>

Bộ công cụ này gồm các chương trình thực thi đáng chú ý sau:

- javac: Chương trình biên dịch các chương trình nguồn viết bằng ngôn ngữ java ra các tập tin thực thi được trên máy ảo Java.
- java: Đây là chương trình làm máy ảo của Java, thông dịch mã Bytecode của các chương trình kiểu application thành mã thực thi của máy thực.
- appletviewer: Bộ thông dịch, thực thi các chương trình kiểu applet.
- javadoc: Tạo tài liệu về chú thích chương trình nguồn một cách tự động.
- jdb: Trình gỡ rối.
- rmic: Tạo Stub cho ứng dụng kiểu RMI.
- rmiregistry: Phục vụ danh bạ (Name Server) trong hệ thống RMI

#### 1.1.6. Kiểu dữ liệu cơ bản dưới Java

- **Kiểu số**

Tên kiểu	Kích thước
byte	1 byte
short	2 bytes
int	4 bytes
long	8 bytes
float	4 bytes
double	8 bytes



- **Kiểu ký tự char**

Java dùng 2 bytes cho kiểu ký tự, theo chuẩn mã UNICODE ( 127 ký tự đầu tương thích với mã ASCII). Do đó, ta sử dụng tương tự như bảng mã ASCII.

- **Kiểu chuỗi ký tự String**

Thực chất đây là một lớp nằm trong thư viện chuẩn của Java (Core API), java.lang.String

- **Kiểu luận lý boolean**

Nhận 2 giá trị là : true và false.

- **Kiểu mảng**

- Khai báo:
  - `int[] a ; float[] yt; String[] names;`
  - hoặc: `int a[]; float yt[]; String names[]; int maTran[][]; float bangDiem[][];`
- Khởi tạo:
  - `a = new int[3]; yt = new float[10]; names = new String[50]; maTran = new int[10][10];`
- Sử dụng mảng:
  - `int i = a[0]; float f = yt[9]; String str = names[20]; int x = maTran[2][5];`

### 1.1.7. Các phép toán cơ bản

Các phép toán trong Java cũng tương tự như trong C++.

- Phép toán số học: `+`, `-`, `*`, `/`, `%`, `++`, `--`, `+=`, `-=`, `*=`, `/=`, `%=`
- Phép toán logic `==`, `!=`, `&&`, `||`, `!`, `>`, `<`, `>=`, `<=`
- Phép toán trên bit : `&`, `|`, `^`, `<<`, `>>`, `~`
- Phép toán điều kiện : `?` :
- Cách chuyển đổi kiểu: (Kiểu Mới)

### 1.1.8. Qui cách đặt tên trong Java

Tên hằng, tên biến, tên lớp, tên phương thức , ... được đặt tên theo qui tắc bắt buộc sau:

- Tên phân biệt giữa chữ hoa và chữ thường.
- Dùng các chữ cái, ký tự số, ký tự `_` và `$`.
- Không bắt đầu bằng ký tự số.
- Không có khoảng trắng trong tên.

Để chương trình nguồn dễ đọc, dễ theo dõi người ta còn sử dụng quy ước đặt tên sau (không bắt buộc):

- Tên lớp:
  - Các ký tự đầu tiên của một từ được viết hoa,
  - Các ký tự còn lại viết thường.
  - Ví dụ: lớp `Nguoi`, `SinhVien`, `MonHoc`, `String`, `InputStream`, `OutputStream`. . .
- Tên biến, tên hằng, tên phương thức:
  - Từ đầu tiên viết thường.
  - Ký tự đầu tiên của từ thứ hai trở đi được viết hoa.
  - Ví dụ: `ten`, `ngaySinh`, `diaChi`, `inTen()`, `inDiaChi()`, `getInputStream()`, . .
- Vị trí đặt dấu `{` và `}` để bắt đầu và kết thúc các khối như sau:

```
if (condition) {  
  
    command1;  
  
    command1;  
  
} else {  
  
    command3;  
  
    command4;  
  
}
```

## 1.2. Chương trình ứng dụng kiểu Application

Java là một ngôn ngữ thuần đối tượng (pure object). Tất cả các thành phần được khai báo như hằng, biến, hàm thủ tục đều phải nằm trong phạm vi của một lớp nào đó. Một ứng dụng trong Java là một tập hợp các lớp liên quan nhau, bao gồm các lớp trong thư viện do Java cung cấp và các lớp được định nghĩa bởi người lập trình. Trong một ứng dụng chỉ có một **Lớp thực thi được**. Đây là lớp đầu tiên được xem xét đến khi chúng ta thực thi ứng dụng.

Lớp thực thi được này có các đặc điểm sau:

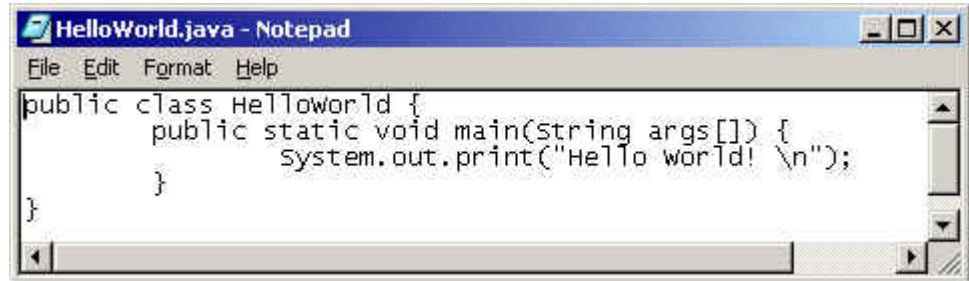
- Có tên lớp trùng với tên tập tin chứa nó.
- Phải khai báo phạm vi là `public`
- Có chứa phương thức:

```
public static void main (String args[]){  
    . . .  
}
```

là phương thức được thực thi đầu tiên.
- Nếu nhiều lớp được định nghĩa trong một tập tin, chỉ có một lớp được khai báo `public`.

### 1.2.1. Chương trình HelloWorld

Trong ví dụ này, chúng ta viết một chương trình ứng dụng in ra màn hình dòng chữ "Hello World !". Đây là ứng dụng đơn giản chỉ có một lớp thực thi được tên là HelloWorld. Lớp này được khai báo là public, có phương thức main(), chứa trong tập tin cùng tên là HelloWorld.java (phần mở rộng bắt buộc phải là .java).



Phương thức System.out.print() sẽ in tất cả các tham số trong dấu () của nó ra màn hình.

Ta có thể dùng bất kỳ chương trình soạn thảo văn bản nào để biên soạn chương trình. Nhưng nhớ phải ghi lại với phần mở rộng là **.java**.

### 1.2.3. Biên soạn chương trình bằng phần mềm Notepad của Ms Windows

Notepad là trình soạn thảo đơn giản có sẵn trong MS Windows mà ta có thể dùng để biên soạn chương trình HelloWorld. Hãy thực hiện các bước sau:

- Chạy chương trình Notepad:
  - Chọn menu Start \ Programs \ Accessories \ Notepad
- Nhập nội dung sau vào Notepad

```
public class HelloWorld {  
    public static void main(String args[]) {  
        System.out.print("Hello World! \n");  
    }  
}
```

- Save tập tin với tên HelloWorld.java
  - Chọn menu File \ Save
  - Tại cửa sổ Save As hãy nhập vào:
    - Save in: Thư mục nơi sẽ lưu tập tin
    - File Name: HelloWorld.java
    - Save as type: All Files
    - Nhấp vào nút Save

#### 1.2.4. Cài đặt bộ phát triển ứng dụng JDK

- Chuẩn bị bộ nguồn cài đặt JDK phù hợp với hệ điều hành sử dụng (Giả sử Windows 2000)
- Chạy tập tin Setup.exe
- Chọn nơi cài đặt, giả sử [D:\jdk1.4](#)
- Đặt biến môi trường
  - PATH = [D:\jdk1.4\bin](#); để có thể thực thi các chương trình này từ bất kỳ thư mục hiện hành nào.
  - CLASSPATH = [D:\jdk1.4\lib](#); chỉ đến các lớp thư viện của Java trong thư mục [D:\jdk1.4\lib](#) và các lớp tại thư mục hiện hành, thể hiện bằng dấu chấm( . ).

#### 1.2.5. Biên dịch và thực thi chương trình

- Mở cửa sổ MS-DOS: Chọn menu Start \ Programs \ Accessories \ Command Prompt.
- Chuyển vào thư mục chứa tập tin HelloWorld.java
- Dùng chương trình javac để biên dịch tập tin HelloWorld.java

```
javac HelloWorld.java
```

- Nếu có lỗi, trên màn hình sẽ xuất hiện thông báo lỗi với dấu ^ chỉ vị trí lỗi.
- Nếu không có lỗi, tập tin thực thi HelloWorld.class được tạo ra.
- Thực thi chương trình HelloWorld.class

```
java HelloWorld
```



Trên màn hình sẽ xuất hiện dòng chữ Hello World!

## 1.2.6. Một số ví dụ

### 1.2.6.1. Hiển thị thông tin ra màn hình

Để in thông tin ra màn hình bạn dùng phương thức

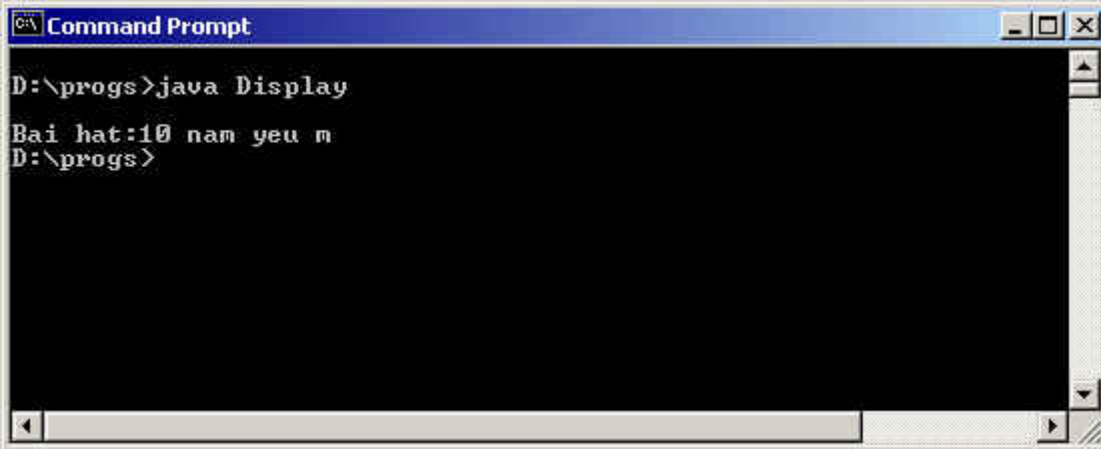
`System.out.print(arg1+ arg2+ .. + argn)`

Java sẽ tự động định dạng dữ liệu cho các tham số `arg1`, `arg2`, ..., `argn` tùy theo kiểu của chúng.

Hãy lưu chương trình sau vào tập tin `Display.java`:

```
public class Display {  
    public static void main(String args[]) {  
        int i = 10;  
        String str = " nam yeu ";  
        char ch = 'm';  
        System.out.print("\n"+ "Bai hat:" + i + str + ch);  
    }  
}
```

Biên dịch và thực thi ta có kết quả :



The screenshot shows a Windows Command Prompt window with the title "C:\> Command Prompt". The command prompt shows the following text:

```
D:\progs>java Display  
Bai hat:10 nam yeu m  
D:\progs>
```

Phương thức `System.out.println(arg1+ arg2+ .. + argn)` in các tham số và tự động xuống dòng mới.

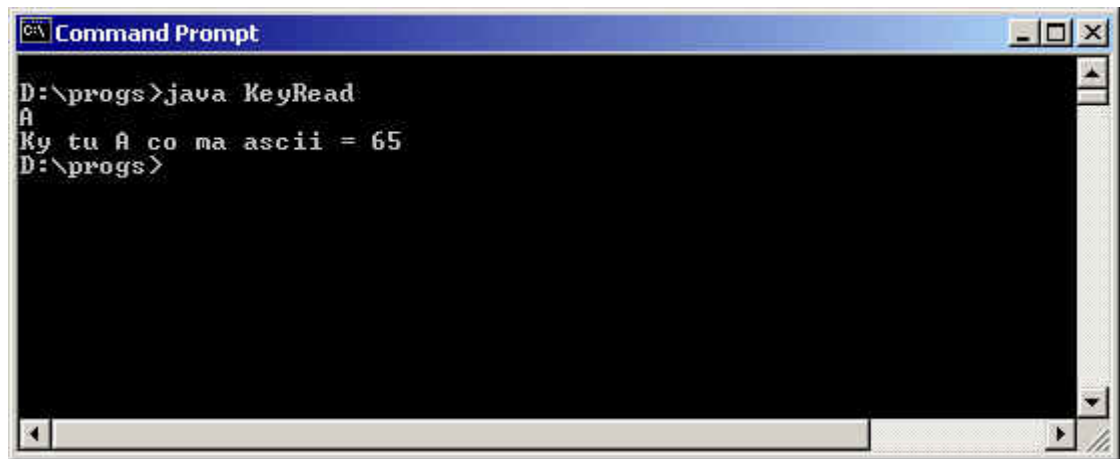
### 1.2.6.2. Đọc ký tự từ bàn phím

Phương thức `int System.in.read()` trả một số nguyên là mã ASCII của ký tự nhập từ bàn phím.

Hãy lưu chương trình sau vào tập tin KeyRead.java

```
import java.io.*;
public class KeyRead {
    public static void main(String args[]) {
        try {
            int ch = System.in.read();
            System.out.print("Ky tu " + (char)ch + " co ma ascii = "+ch);
        } catch(IOException ie) {
            System.out.print("Error " + ie);
        }
    }
}
```

Biên dịch và thực thi ta có kết quả :



Trong ví dụ trên lưu ý một số điểm sau:

- Dòng đầu tiên `import java.io.*;` là cơ chế để khai báo với trình biên dịch các lớp thư viện của Java mà chương trình có sử dụng đến. Trong trường hợp này chương trình khai báo sử dụng tất cả các lớp trong gói (package) `java.io`. Thực tế chương trình trên chỉ sử dụng lớp `IOException` của gói `java.io` mà thôi, vì thế ta có thể thay thế dòng `java.io.*;` bằng `java.io.IOException;`.
- Cơ chế ngoại lệ (Exception) của java:

```
try {
    ....
} catch(IOException ie) {
    ....
}
```

sẽ được giải thích rõ ở phần sau.

## 1.3. Các cấu trúc điều khiển trong Java

### 1.3.1. Lệnh if – else

#### Cú pháp:

```
if (Condition) {  
    // Các lệnh sẽ được thực hiện nếu giá trị của Condition là true  
}
```

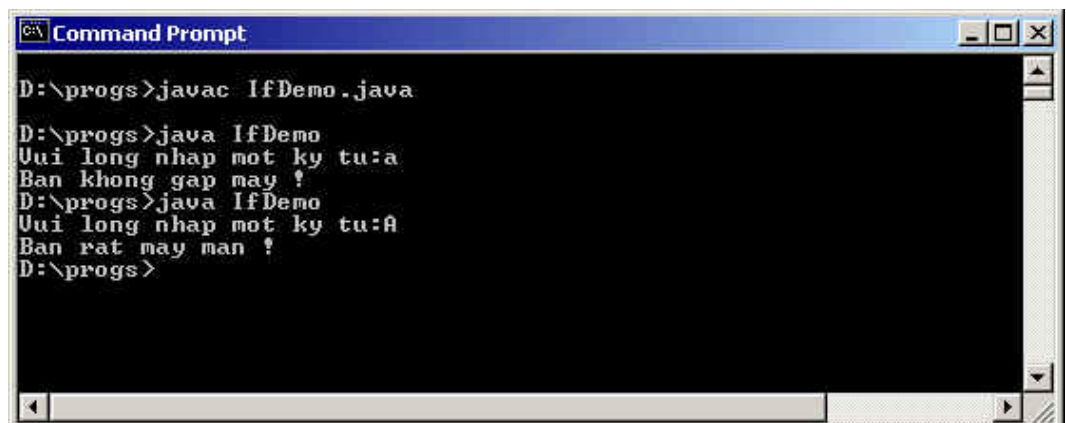
```
if (Condition) {  
    // Các lệnh sẽ được thực hiện nếu giá trị của Condition là true  
} else {  
    // Các lệnh sẽ được thực hiện nếu giá trị của Condition là false  
}
```

#### Ví dụ:

Lưu chương trình sau vào tập tin IfDemo.java :

```
import java.io.*;  
public class IfDemo {  
    public static void main(String args[]) {  
        System.out.print("Vui long nhap mot ky tu:");  
        try {  
            int ch = System.in.read();  
            if (ch == 'A') {  
                System.out.print("Ban rat may man !");  
            }  
            else {  
                System.out.print("Ban khong gap may !");  
            }  
        } catch (IOException ie) {  
            System.out.print("Error:"+ie);  
        }  
    }  
}
```

Biên dịch và thực thi có kết quả như sau:



```
Command Prompt  
D:\progs>javac IfDemo.java  
D:\progs>java IfDemo  
Vui long nhap mot ky tu:a  
Ban khong gap may !  
D:\progs>java IfDemo  
Vui long nhap mot ky tu:A  
Ban rat may man !  
D:\progs>
```

### 1.3.2. Phép toán ?

#### Cú pháp:

(condition) ? Operation1 : Operation2;


Nếu điều kiện condition có giá trị là true lệnh sẽ trả về giá trị của biểu thức Operation1, ngược lại sẽ trả về giá trị của biểu thức Operation2.

#### Ví dụ:

Lưu chương trình sau vào tập tin QuestionOp.java :

```
import java.io.*;
public class QuestionOp {
    public static void main(String args[]) {
        System.out.print("Vui long nhap mot ky tu:");
        try {
            int ch = System.in.read();
            int point = (ch == 'A') ? 10:0;
            System.out.print("Diem cua ban la:"+point);
        } catch(IOException ie) {
            System.out.print("Error:"+ ie);
        }
    }
}
```

Biên dịch và thực thi được kết quả như sau:



```
Command Prompt
D:\progs>javac QuestionOp.java
D:\progs>java QuestionOp
Vui long nhap mot ky tu:a
Your point is 0
D:\progs>java QuestionOp
Vui long nhap mot ky tu:A
Your point is 10
D:\progs>_
```



### 1.3.3. Lệnh switch

#### Cú pháp

```
switch ( variable ) {  
    case value1 : {  
        Task 1;  
        // Các tác vụ sẽ được thực thi nếu giá trị của variable là value1  
        break;  
    }  
    case value2 : {  
        Task 2;  
        // Các tác vụ sẽ được thực thi nếu giá trị của variable là value2  
        break;  
    }  
    ...  
    default:  
        Task n;  
        // Tác vụ sẽ được thực thi nếu giá trị của variable không là các giá trị trên  
}
```

#### Ví dụ

Lưu chương trình sau vào tập tin CaseOp.java

```
import java.io.*;  
public class CaseOp {  
    public static void main(String args[]) {  
        System.out.print("Enter a number character: ");  
        try {  
            int ch = System.in.read();  
            switch(ch) {  
                case '0': { System.out.print("Zero");break;}  
                case '1': { System.out.print("One"); break;}  
                case '2': { System.out.print("Two"); break;}  
                case '3': { System.out.print("Three");break;}  
                case '4': { System.out.print("Four"); break;}  
                case '5': { System.out.print("Five"); break;}  
                case '6': { System.out.print("Six"); break;}  
                case '7': { System.out.print("Seven");break;}  
                case '8': { System.out.print("Eight");break;}  
                case '9': { System.out.print("Nine"); break;}  
                default: { System.out.print("I don't know"); break;}  
            }  
        } catch(IOException ie) {  
            System.out.print("Error "+ie);  
        }  
    }  
}
```

Biên dịch và thực thi được kết quả sau:



```
Command Prompt

D:\progs>javac CaseOp.java

D:\progs>java CaseOp
Enter a number character: 9
Nine
D:\progs>java CaseOp
Enter a number character: a
I don't know
D:\progs>_
```

### 1.3.4. Lệnh while

#### Cú pháp


```
while (condition) {
    // nếu condition có giá trị là true, thì các tác vụ ở đây sẽ được lặp lại
}
```

#### Ví dụ

Lưu chương trình sau vào tập tin WhileDemo.java

```
import java.io.*;
public class WhileDemo {
    public static void main(String args[]) {
        int num = '9';
        while (num > '0') {
            System.out.print((char)num + " ");
            num--;
        }
    }
}
```

Biên dịch và thực thi được kết quả sau:



```
Command Prompt

D:\progs>javac WhileDemo.java

D:\progs>java WhileDemo
9 8 7 6 5 4 3 2 1
D:\progs>
```

### 1.3.5. Lệnh do - while

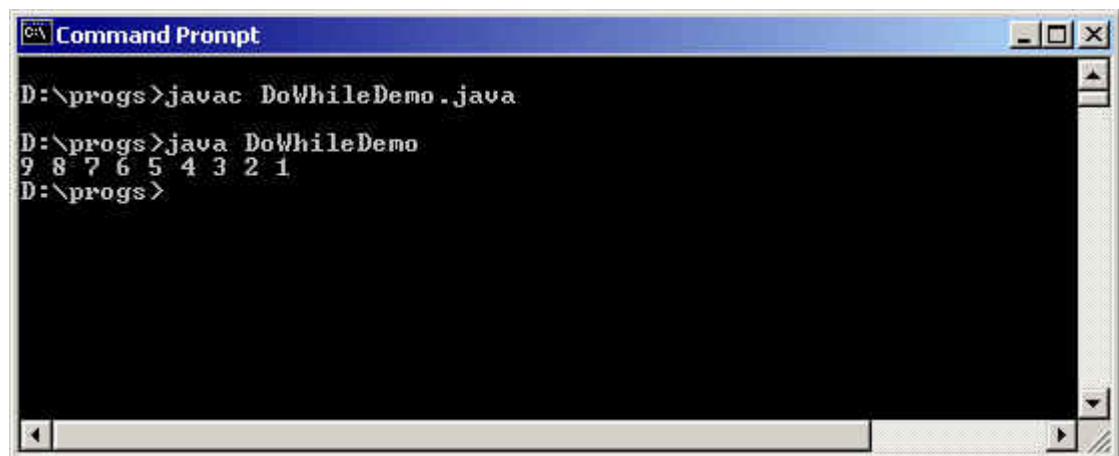
#### Cú pháp

```
do {  
    // Lặp lại các tác vụ ở đây cho đến khi điều kiện condition có giá trị là false  
} while (condition)
```

**Ví dụ:** Lưu chương trình sau vào tập tin DoWhileDemo.java

```
import java.io.*;  
public class DoWhileDemo {  
    public static void main(String args[]) {  
        int num = '9';  
        do {  
            System.out.print((char)num + " ");  
            num--;  
        } while (num > '0');  
    }  
}
```

Biên dịch và thực thi được kết quả sau:



```
C:\ Command Prompt  
D:\progs>javac DoWhileDemo.java  
D:\progs>java DoWhileDemo  
9 8 7 6 5 4 3 2 1  
D:\progs>
```

### 1.3.6. Lệnh for

#### Cú pháp

```
for (operation1; condition; operation2){  
    // Các tác vụ được lặp lại  
}
```

Tương đương như cấu trúc sau:

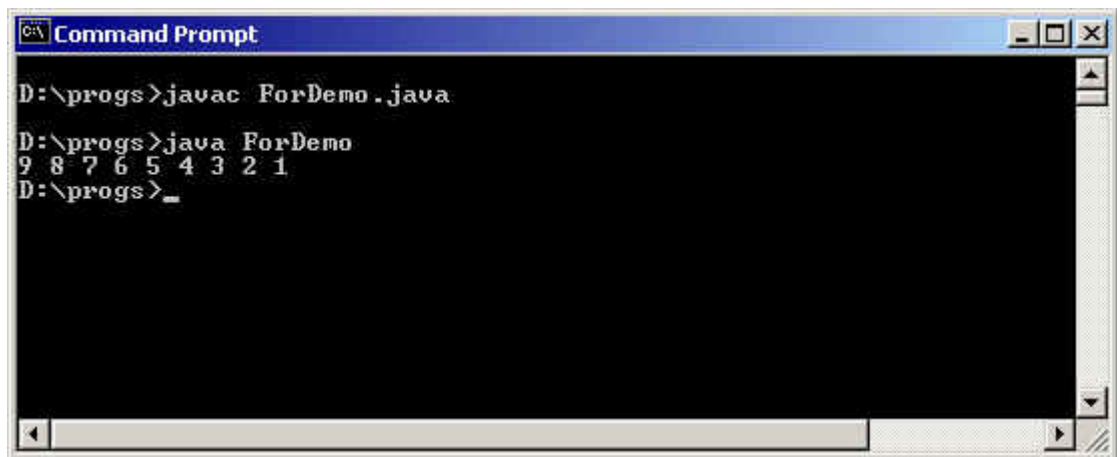
```
operation1;  
while (condition) {  
    // Các tác vụ được lặp lại  
    operation2;  
}
```

#### Ví dụ

Lưu chương trình sau vào tập tin ForDemo.java

```
import java.io.*;
public class ForDemo {
    public static void main(String args[]) {
        for(int num = '9'; num>'0'; num --) {
            System.out.print((char)num + " ");
        }
    }
}
```

Biên dịch và thực thi được kết quả như sau:



```
C:\> Command Prompt

D:\progs>javac ForDemo.java

D:\progs>java ForDemo
9 8 7 6 5 4 3 2 1
D:\progs>_
```

### 1.3.7. Lệnh break

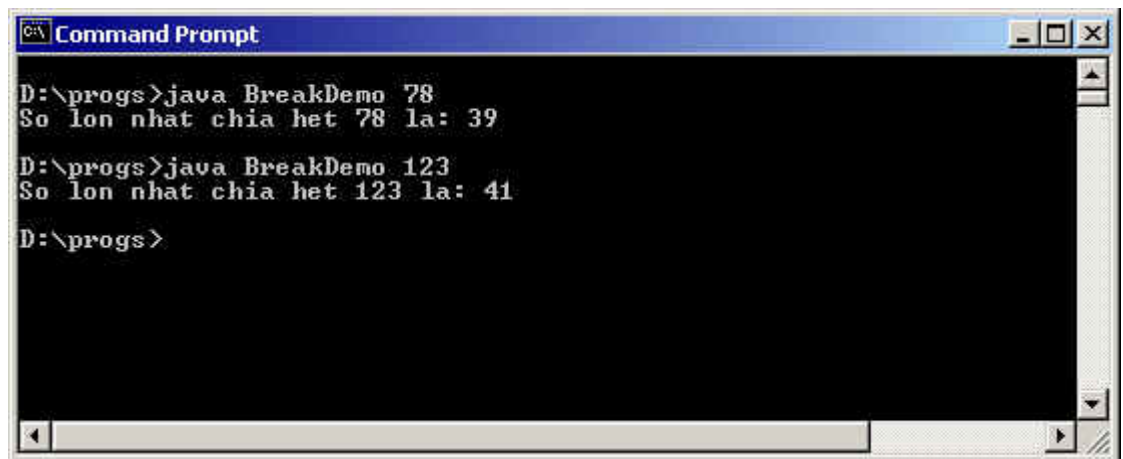
Vòng lặp của các lệnh while, do-while và for sẽ kết thúc khi lệnh break được thực hiện.

#### Ví dụ

Lưu chương trình sau vào tập tin BreakDemo.java

```
import java.io.*;
public class BreakDemo {
    public static void main(String args[]){
        int num =Integer.valueOf(args[0]).intValue();
        int i= num / 2;
        while(true){
            if (num % i ==0) break;
            i--;
        }
        System.out.println("So lon nhat chia het "+num+ " la: "+i);
    }
}
```

Biên dịch và thực thi được kết quả sau:



```

C:\> Command Prompt

D:\progs>java BreakDemo 78
Số lớn nhất chia hết 78 là: 39

D:\progs>java BreakDemo 123
Số lớn nhất chia hết 123 là: 41

D:\progs>

```

Chương trình trên đổi đổi số thứ nhất của nó (lưu trong args[0]) thành số ( bằng lệnh Integer.valueOf(args[0]).intValue() ) và tìm số lớn nhất chia hết số này.

### 1.3.8. Lệnh continue

Trong một **lần lặp** nào đó của các lệnh while, do-while và for, nếu gặp lệnh continue thì **lần lặp** sẽ kết thúc (bỏ qua các lệnh phía sau continue) để bắt đầu lần lặp tiếp theo.


**Ví dụ:** Lưu chương trình sau vào tập tin ContinueDemo.java

```

import java.io.*;
public class ContinueDemo{
    public static void main(String args[]){
        int num =Integer.valueOf(args[0]).intValue();
        System.out.print("The odd numbers: ");
        for (int i =0; i< num; i++ ){
            if (i % 2 ==0) continue;
            System.out.print(i+ " ");
        }
    }
}

```

Biên dịch và thực thi được kết quả sau:



```

C:\> Command Prompt

D:\progs>javac ContinueDemo.java

D:\progs>java ContinueDemo 21
The odd numbers: 1 3 5 7 9 11 13 15 17 19

D:\progs>_

```

Chương trình này in ra tất cả các số lẻ nhỏ hơn số đưa vào từ đối số.

### 1.3.9. Một số vấn đề khác

#### 1.3.9.1. Đọc đối số của chương trình

Khi thực thi chương trình ta có thể nhập vào các đối số từ dòng lệnh theo cú pháp sau:

```
java ClassName arg1 arg2 arg3 argn
```

Các đối số cách nhau khoảng trắng. Để đón nhận các đối số này, phương thức main bắt buộc phải khai báo một tham số kiểu mảng các chuỗi

```
public static void main(String args[]) {  
    ...  
}
```

Các đối số lần lượt được đặt vào các phần tử của mảng này. Số lượng đối số có thể xác định được bằng cách truy xuất thuộc tính **args.length** của mảng.

#### Ví dụ

Lưu chương trình sau vào tập tin PrintArgs.java

```
public class PrintArgs {  
    public static void main (String args[]) {  
        for (int i = 0; i < args.length; i++) {  
            System.out.println(args[i]);  
        }  
    }  
}
```

Biên dịch và thực thi chương trình được kết quả sau:



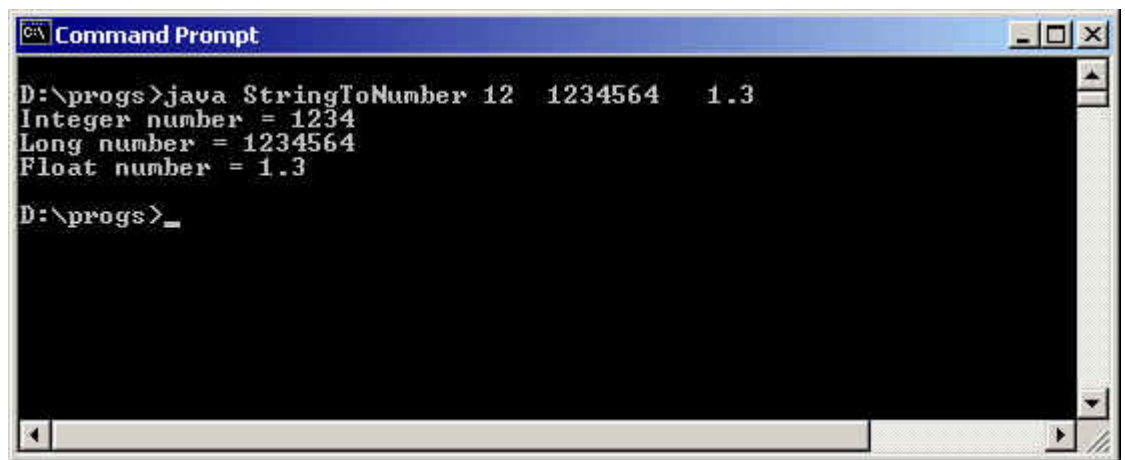
```
Command Prompt  
D:\progs>java PrintArgs arg1 arg2 1234  
arg1  
arg2  
1234  
D:\progs>
```

### 1.3.9.2. Đổi chuỗi thành số

Lưu chương trình sau vào tập tin StringToNumber.java

```
public class StringToNumber{
    public static void main (String args[]) {
        int i = Integer.valueOf( args[0]).intValue();
        long l = Long.valueOf( args[1]).longValue();
        float f = Float.valueOf( args[2]).floatValue();
        System.out.println("Integer number = "+i );
        System.out.println("Long number = "+l );
        System.out.println("Float number = "+f );
    }
}
```

Biên dịch và thực thi chương trình được kết quả sau:



```
Command Prompt
D:\progs>java StringToNumber 12 1234564 1.3
Integer number = 1234
Long number = 1234564
Float number = 1.3
D:\progs>_
```

## 1.4. Ngoại lệ (EXCEPTION)

Trong chương trình, có một số các "thao tác không chắc chắn", ví dụ như các thao tác vào/ra: đĩa mềm chưa sẵn sàng, máy in có lỗi, nối kết mạng không thực hiện được . . . sẽ dẫn đến lỗi thực thi chương trình.

Java hạn chế các lỗi sinh ra từ "thao tác không chắc chắn" bằng cơ chế Ngoại lệ (Exception).

Ngoại lệ tức là một sự kiện xảy ra ngoài dự tính của chương trình nếu không xử lý sẽ làm cho chương trình chuyển sang trạng thái không còn kiểm soát được. Ví dụ điều gì sẽ xảy ra nếu chương trình truy xuất đến phần tử thứ 11 của một mảng 10 phần tử ? Một số ngôn ngữ như C, C++ sẽ không báo lỗi gì cả, chương trình vẫn tiếp tục vận hành nhưng kết quả thì không thể xác định được.

Để hạn chế những lỗi như thế, Java bắt buộc các lệnh có thể dẫn đến các ngoại lệ phải có các đoạn mã xử lý phòng hờ khi ngoại lệ xảy ra theo cú pháp sau:

```
try {  
    Các thao tác vào ra có thể sinh ra các ngoại lệ.  
}  
catch (KiểuNgoạiLệ_01 biến) {  
    ứng xử khi ngoại lệ KiểuNgoạiLệ_01 sinh ra  
}  
catch (KiểuNgoạiLệ_02 biến) {  
    ứng xử khi ngoại lệ KiểuNgoạiLệ_02 sinh ra  
}  
finally { Công việc luôn luôn được thực hiện }
```

Trong cơ chế này, các lệnh có thể tạo ra ngoại lệ sẽ được đưa vào trong khối bao bọc bởi từ khóa **try {}**. Tiếp theo đó là một loạt các khối **catch {}**. Một lệnh có thể sinh ra một hoặc nhiều loại ngoại lệ. Ứng với một loại ngoại lệ sẽ có một khối **catch {}** để xử lý cho loại ngoại lệ đó. Tham số của **catch** chỉ ra loại ngoại lệ mà nó có trách nhiệm xử lý. Khi thực thi chương trình, nếu một lệnh nào đó nằm trong khối **try {}** tạo ra ngoại lệ, điều khiển sẽ được chuyển sang các lệnh nằm trong các khối **catch {}** tương ứng với loại ngoại lệ đó. Các lệnh phía sau lệnh tạo ra ngoại lệ trong khối **try {}** sẽ bị bỏ qua. Các lệnh nằm trong khối **finally {}** thì luôn luôn được thực hiện cho dù có xảy ra ngoại lệ hay là không. Khối lệnh **finally {}** là tùy chọn có thể không cần.

Ngoại lệ có loại bắt buộc phải xử lý, tức phải có **try {}**, có **catch {}** khi sử dụng lệnh đó. Ví dụ như lệnh đọc từ bàn phím. Trình biên dịch của java sẽ báo lỗi nếu chúng ta không xử lý chúng.

Ngược lại, có loại ngoại lệ không bắt buộc phải xử lý, ví dụ như truy xuất đến phần tử bên ngoài chỉ số mảng.

Tra cứu tài liệu đặc tả các API của java để biết được các ngoại lệ tạo ra từ một phương thức.

### Ví dụ:

Lưu chương trình sau vào tập tin `ExceptionDemo.java` :

```
public class ExceptionDemo {  
    public static void main(String[] args) {  
        try {  
            System.out.println("Hello " + args[0]);  
        }  
        catch (ArrayIndexOutOfBoundsException e){  
            System.out.println("Hello Whoever you are.");  
        }  
        finally {  
            System.out.println("How are you?");  
        }  
    }  
}
```

Biên dịch và thực thi có kết quả như sau:





```
Command Prompt
D:\progs>javac ExceptionDemo.java
D:\progs>java ExceptionDemo
Hello Whoever you are.
How are you?
D:\progs>java ExceptionDemo Hung
Hello Hung
How are you?
D:\progs>
```

Trong chương trình trên chúng ta dự định sẽ chào người được đưa vào từ đối số thứ nhất của chương trình (được chứa trong phần tử `args[0]`). Tuy nhiên nếu người dùng thực thi chương trình quên đưa vào đối số, tức phần tử `args[0]` không tồn tại. Ngoại lệ báo hiệu truy xuất đến phần tử nằm ngoài mảng (`ArrayIndexOutOfBoundsException`) được quăng ra (throw). Khi đó đoạn mã lệnh trong khối catch có tham số là loại ngoại lệ `ArrayIndexOutOfBoundsException` sẽ được thực hiện.

## 1.5. Một số vấn đề liên quan đến lớp trong Java

### 1.5.1. Định nghĩa lớp mới

Ngoài các lớp được định nghĩa sẵn trong thư viện chuẩn của java, các lập trình viên có thể định nghĩa thêm các lớp của mình theo cú pháp sau:

```
class ClassName {
    // Danh sách các thuộc tính thuộc lớp
    DataType01 attribute1, attribute2, . . . ;
    DataType02 attribute3, attribute4, . . . ;
    // Danh sách các phương thức thuộc lớp
    ClassName([DataType parameter, DataType parameter]) {
        // Constructor
        ...
    }
    void method01() {
        ...
    }
    DataType method02( . . . ) {
        ...
        return xx;
    }
}
```

`ClassName` là tên lớp mới đang được định nghĩa.

Tạo đối tượng tên obj thuộc lớp `ClassName`.

`ClassName obj = new ClassName();`

Ví dụ: Định nghĩa một lớp có:

- Tên là Person
- Hai thuộc tính là name và address
- Phương thức khởi tạo có hai tham số để gán giá trị khởi động cho hai thuộc tính.
- Phương thức void display() cho biết người đó tên là gì, địa chỉ ở đâu.
- Phương thức main() tạo ra một đối tượng tên là tom thuộc lớp Person

Lưu chương trình sau vào tập tin Person.java

```
public class Person{
    String name;    //Thuộc tính
    String address; //Thuộc tính
    Person(String n, String address) {    // Phương thức khởi tạo
        name = n;
        this.address = address;
    }
    void display(){                // Hiển thị tên và địa chỉ
        System.out.print(name + " is at "+ address);
    }
    public static void main(String args[]){
        Person tom = new Person("Tom","Disney Land"); // Tạo đối tượng
        tom.display(); // Gọi phương thức của đối tượng
    }
}
```

Biên dịch và thực thi ta được kết quả:



```
Command Prompt
D:\progs>javac Person.java
D:\progs>java Person
Tom is at Disney Land
D:\progs>
```

### 1.5.2. Phạm vi nhìn thấy của một lớp

Một lớp được định nghĩa và cài đặt bên trong một tập tin. Một tập tin có thể chứa một hoặc nhiều lớp. Trong một tập tin, chỉ có một lớp được khai báo là **public** (phía trước từ khóa class), các lớp còn lại phải là private (mặc nhiên). Một lớp được khai báo là public sẽ được nhìn thấy bởi các lớp khác ở cùng tập tin hay khác tập tin với nó. Ngược lại các lớp private chỉ được nhìn thấy bởi các lớp nằm cùng tập tin với nó mà thôi.

**Ví dụ:** Trong ví dụ này, chúng ta tách phương thức main ra khỏi lớp Person và đưa nó vào lớp mới MultiClass. Lưu hai lớp này vào trong cùng một tập tin tên là MultiClass.java, với lớp MultiClass được khai báo là public, lớp Person khai báo private.

```
// Lớp có phạm vi public có thể tham khảo từ bên ngoài tập tin
public class MultiClass {
    public static void main(String args[]){
        Person tom = new Person("Tom","Disney Land");
        tom.display();
    }
}
// Lớp có phạm vi private chỉ có thể tham khảo bởi các lớp nằm cùng tập tin
class Person{
    String name;
    String address;

    Person(String n, String address) {
        name = n;
        this.address = address;
    }
    void display(){
        System.out.println(name + " is at "+ address);
    }
}
```

Biên dịch và thực thi ta được kết quả:



```
Command Prompt
D:\progs>javac MultiClass.java
D:\progs>java MultiClass
Tom is at Disney Land
D:\progs>_
```

### 1.5.3. Tính thừa kế

- Một lớp chỉ có thể có một lớp cha (thừa kế đơn).
- Lớp cha được tham khảo từ lớp con bởi từ khóa **super**.
- Dùng từ khóa **extends** để khai báo thừa kế.

Cú pháp:

```
class A extends B { // Khai báo A thừa kế từ B
    ...
}
```

Ví dụ: Định nghĩa lớp Client có các đặc điểm sau:

- Thừa kế từ lớp Person.
- Có thêm thuộc tính: telephone và buy (lượng hàng mua).
- Có phương thức khởi tạo.
- Định nghĩa lại phương thức void display() của lớp cha.

Lưu chương trình sau vào tập tin Client.java

```
public class Client extends Person {
    int telephone;
    long buy;
    public Client(String n, String a, int t, long b) {
        super(n,a);
        telephone=t;
        buy=b;
    }
    public void display() {
        super.display();
        System.out.println( " , Number of telephone:"+ telephone + " , buy: "+
buy );
    }
    public static void main(String args[]){
        Client tom = new Client("Tom","Disney Land",123456,1000);
        tom.display();
    }
}
```

Biên dịch và thực thi ta được kết quả:



```
Command Prompt

D:\progs>javac Client.java

D:\progs>java Client
Tom is at Disney Land
 , Number of telephone:123456 , buy: 1000

D:\progs>_
```

## 1.6. Vào / Ra với Stream

Stream là một dòng liên tục, có thứ tự các bytes dữ liệu chảy giữa chương trình và các thiết bị ngoại vi. Nó là khái niệm trừu tượng giúp giảm bớt các thao tác vào ra phức tạp đối với người lập trình. Nó cho phép nối kết nhiều loại thiết bị ngoại vi khác nhau với chương trình.

Nếu dòng dữ liệu trong Stream có hướng chảy từ thiết bị ngoại vi vào chương trình thì ta nói đây là Stream nhập (Input Stream), ngược lại là Stream xuất (Output Stream).

Đối với Java, các thiết bị chỉ nhập, như bàn phím, sẽ có các Stream nhập nối với nó, các thiết bị chỉ xuất, như màn hình, sẽ có các stream xuất nối với nó , các thiết bị vừa xuất, vừa nhập, như đĩa từ, thì có cả stream nhập và xuất nối với nó.

Để giao tiếp với các thiết bị ngoại vi, chương trình trước tiên phải lấy được các stream nhập / xuất gắn với thiết bị ngoại vi này. Sau đó, chương trình có thể gởi dữ liệu ra ngoại vi bằng thao tác ghi vào Stream xuất của ngoại vi. Ngược lại, chương trình có thể nhận dữ liệu từ ngoại vi bằng thao tác đọc stream nhập của ngoại vi đó.

Như vậy, chương trình chỉ làm việc trên các stream nhập và stream xuất, mà không quan tâm đến đặc điểm riêng biệt của thiết bị ngoại vi nối với Stream. Điều này giúp chương trình giao tiếp với hệ thống mạng cũng dễ dàng như giao tiếp với màn hình, bàn phím hay đĩa từ.

Một điểm khác cần lưu ý là stream bao gồm những bytes rời rạc. Những bytes này mô tả những dạng dữ liệu khác nhau. Ví dụ một số integer khi viết vào stream sẽ chuyển thành 4 bytes. Vì thế cần phải có các thao tác chuyển đổi dữ liệu nhận và gởi giữa chương trình và stream.

Java hỗ trợ hai các lớp stream cơ bản trong gói java.io là:

- java.io.InputStream: Stream nhập
- java.io.OutputStream: Stream xuất

Ngoài ra còn có các lớp Stream thừa kế từ hai lớp trên nhằm mục đích cung cấp các tiện ích cho các loại thiết bị vào ra chuyên biệt như: FileInputStream, FileOutputStream, PipedInputStream, PipedOutputStream, . . .

### 1.6.1. Lớp java.io.InputStream

Là loại stream cho phép chương trình nhận dữ liệu từ ngoại vi. Có các phương thức cơ bản sau:

#### **int read() throws IOException :**

Đọc 1 byte từ Stream

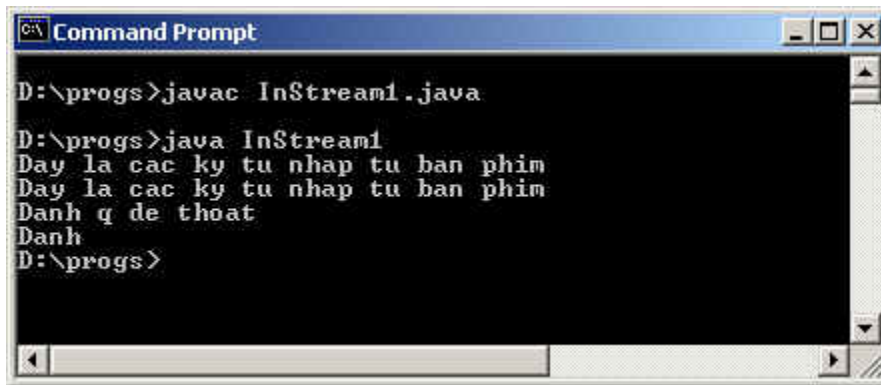
- Return 0-255 : Mã ASCII của byte nhận được từ ngoại vi
- -1 : Stream đã kết thúc, không còn dữ liệu.

Đối với Java, System.in là một InputStream nối kết với bàn phím được tạo sẵn bởi hệ thống. Chương trình có thể dùng InputStream này để nhận các ký tự nhập từ bàn phím.

Ví dụ: Hãy lưu chương trình sau vào tập tin InStream1.java

```
import java.io.*;
public class InStream1 {
    public static void main(String args[]) {
        InputStream is = System.in; // KeyBoard = System.in
        while (true) {
            try {
                int ch = is.read();
                if (ch == -1 || ch == 'q') break;
                System.out.print((char)ch);
            } catch (IOException ie) {
                System.out.print("Error: "+ie);
            }
        }
    }
}
```

Biên dịch và thực thi ta được kết quả sau:



```
Command Prompt
D:\progs>javac InStream1.java
D:\progs>java InStream1
Day la cac ky tu nhap tu ban phim
Day la cac ky tu nhap tu ban phim
Danh q de thoát
Danh
D:\progs>
```

Ví dụ trên **chờ** nhận các ký tự được nhập từ bàn phím.

### **int read(byte b[]) throws IOException:**

Đọc tất cả các byte hiện có trong Stream vào mảng b.

- Return 0-255: Số lượng byte đọc được.
- -1 : Stream đã kết thúc, không còn dữ liệu.

### **int read(byte b[], int offset, int len)**

Đọc len byte từ Stream hiện tại, lưu vào trong mảng b bắt đầu từ vị trí offset

- Return: số lượng byte đọc được.
- -1 : Stream đã kết thúc.

Các phương thức trên khi thực thi sẽ bị nghẽn (block) cho đến khi có dữ liệu hoặc kết thúc Stream hay một ngoại lệ xuất hiện.

### **int available()**

Trả về số lượng byte hiện có trong Stream mà không làm nghẽn chương trình.

#### **Ví dụ:**

Lưu chương trình sau vào tập tin có tên InStream2.java

```
import java.io.*;
public class InStream2 {
    public static void main(String args[]) {
        InputStream is = System.in; // KeyBoard = System.in
        while (true) {
            try {
                int num = is.available();
                if (num > 0){
                    byte[] b = new byte[num];
                    int result = is.read(b);
                    if (result == -1) break;
                    String s = new String(b);
                    System.out.print(s);
                } else {
```

```

        System.out.print('.');
    }
} catch (IOException ie) {
    System.out.print("Error: "+ie);
}
}
}
}

```

Biên dịch và thực thi ta được kết quả sau:



Điểm khác biệt trong ví dụ này là các ký tự ta nhập từ bàn phím sẽ không hiển thị tức thì trên màn hình. Chúng chỉ hiển thị sau khi chúng ta nhấn phím Enter.

### 1.6.2. Lớp `java.io.OutputStream`

Là loại stream cho phép chương trình xuất dữ liệu ra ngoại vi. Có các phương thức cơ bản sau:

#### **`void write(int b) throws IOException`**

- Viết byte `b` vào Stream hiện tại,
- Return : void

#### **`void write (byte[] b) throws IOException`**

- Viết tất cả các phần tử của mảng `b` vào Stream hiện tại
- Return : void

#### **`void write (byte[] b, int offset, int len) throws IOException:`**

- Viết `len` phần tử trong mảng `b` vào Stream hiện tại, bắt đầu từ phần tử có chỉ số là `offset` của mảng.
- Return : void

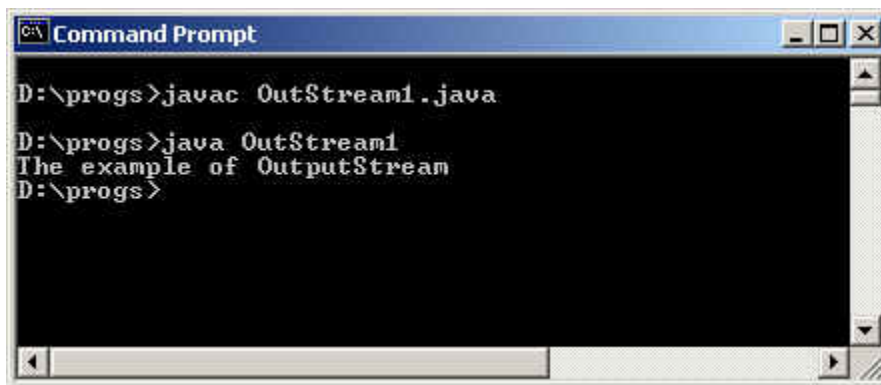
Đối với Java, **`System.out`** là một `OutputStream` nối kết với màn hình được tạo sẵn bởi hệ thống. Chương trình có thể dùng `OutputStream` này để gửi các ký tự ra màn hình.

**Ví dụ:**

Hãy lưu chương trình sau vào tập tin OutputStream1.java

```
import java.io.*;
public class OutputStream1 {
    public static void main(String args[]) {
        OutputStream os = System.out; // Monitor = System.out
        try {
            String str = "The example of OutputStream";
            byte b[] = str.getBytes(); // Đổi chuỗi thành mảng các bytes
            os.write(b);
        } catch (IOException ie) {
            System.out.print("Error: "+ie);
        }
    }
}
```

Biên dịch và thực thi chương trình ta được kết quả sau:



### 1.6.3. Nhập chuỗi từ một InputStream

InputStream là Stream nhập gồm chuỗi các bytes. Nó chỉ cung cấp các phương thức cho việc đọc byte và mảng các bytes. Để có thể đọc được chuỗi từ một InputStream ta phải sử dụng thêm các lớp sau:

- Lớp **java.io.InputStreamReader**: Là cầu nối để chuyển InputStream dạng byte sang InputStream dạng các ký tự (Character).
- Lớp **java.io.BufferedReader**: Hỗ trợ việc đọc văn bản từ một InputStream dạng ký tự.

Phương thức **String readLine() throws IOException** của **BufferedReader** cho phép đọc dòng văn bản kế tiếp trong InputStream. Một dòng kết thúc bởi cặp ký tự '\r'\n' hoặc kết thúc Stream.

Return: Một chuỗi ký tự hoặc null.

Giả sử **is** là một đối tượng thuộc lớp InputStream. Để đọc chuỗi từ is ta thực hiện các thao tác sau:

1. `InputStreamReader isr = new InputStreamReader(is);`



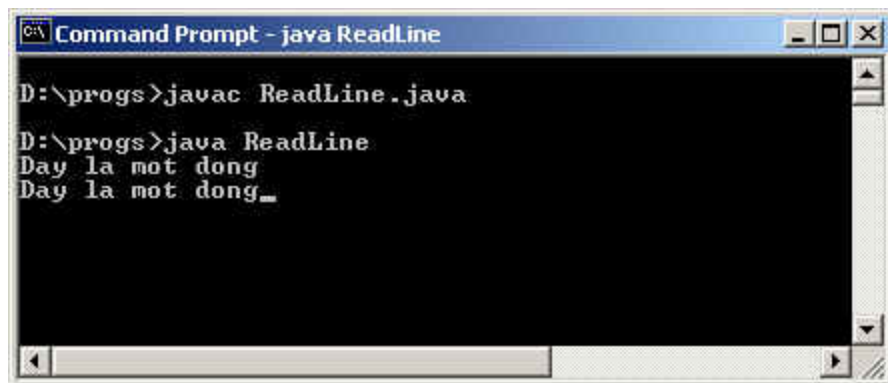
2. `BufferedReader br = new BufferedReader (isr);`
3. `String str = br.readLine();`

### Ví dụ: Đọc chuỗi từ bàn phím

Lưu chương trình sau vào tập tin `ReadLine.java`

```
import java.io.*;
public class ReadLine{
    public static void main(String args[]) {
        InputStreamReader isr = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(isr);
        while (true) {
            try {
                String line = br.readLine();
                if (line == null ) break;
                System.out.print(line);
            } catch (IOException ie) {
                System.out.print("Error: "+ie);
            }
        }
    }
}
```

Biên dịch và thực thi ta có kết quả sau:



### 1.6.4. Xuất chuỗi ra một `OutputStream`

`OutputStream` là Stream xuất gồm chuỗi các bytes. Nó chỉ cung cấp các phương thức cho việc viết byte và mảng các bytes. Để có thể gửi được chuỗi ra một `OutputStream` ta phải sử dụng lớp `java.io.PrintWriter`.

Giả sử: `os` là một `OutputStream`, `str` là chuỗi cần viết vào `os`.

Ta thực hiện các thao tác sau:

1. `PrintWriter pw = new PrintWriter(os);`
2. `pw.wirte(str);`
3. hoặc `pw.println(str);` // Nếu muốn có ký tự xuống dòng
4. `flush()` // Đẩy dữ liệu từ buffer ra ngoài vì

### Ví dụ: Viết chuỗi ra màn hình

Lưu chương trình sau vào tập tin PrintString.java

```
import java.io.*;
public class PrintString {
    public static void main(String args[]) {
        OutputStream os = System.out;
        PrintWriter pw = new PrintWriter(os);
        pw.write("This is a string \r\n");
        pw.println("This is a line");
        pw.write("Bye! Bye!");
        pw.flush();
    }
}
```

Biên dịch và thực thi ta được:



```

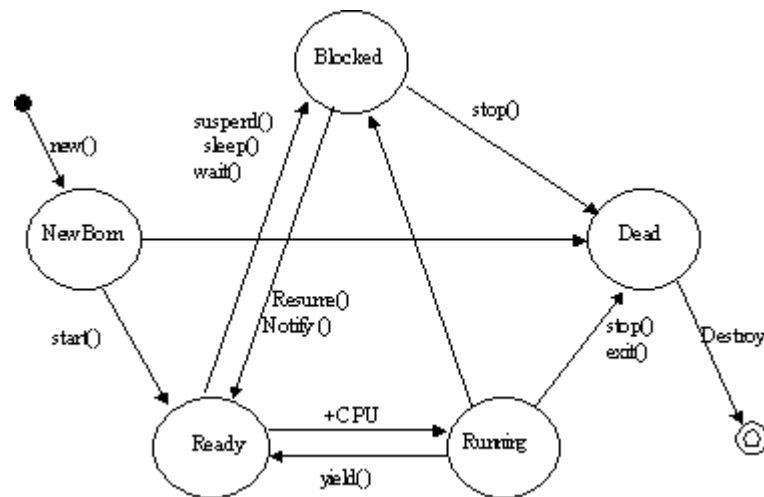
C:\ Command Prompt
D:\progs>javac PrintString.java
D:\progs>java PrintString
This is a string
This is a line
Bye! Bye!
D:\progs>
```

## 1.7. Luồng (Thread)

Luồng là một cách thông dụng để nâng cao năng lực xử lý của các ứng dụng nhờ vào cơ chế song song. Trong một hệ điều hành cổ điển, đơn vị cơ bản sử dụng CPU là một quá trình. Mỗi quá trình có một Thanh ghi bộ đếm chương trình (PC-Program Counter), Thanh ghi trạng thái (Status Register), ngăn xếp (Stack) và không gian địa chỉ riêng (Address Space).

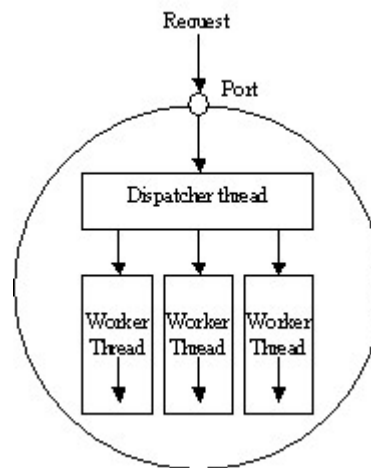
Ngược lại, trong một hệ điều hành có hỗ trợ tiện ích luồng, đơn vị cơ bản sử dụng CPU là một luồng. Trong những hệ điều hành này, một quá trình bao gồm một không gian địa chỉ và nhiều luồng điều khiển. Mỗi luồng có bộ đếm chương trình, trạng thái thanh ghi và ngăn xếp riêng. Nhưng tất cả các luồng của một quá trình cùng chia sẻ nhau một không gian địa chỉ. Nhờ đó các luồng có thể sử dụng các biến toàn cục, chia sẻ các tài nguyên như tập tin, hiệu báo một cách dễ dàng...

Cách thức các luồng chia sẻ CPU cũng giống như cách thức của các quá trình. Một luồng cũng có những trạng thái: đang chạy (running), sẵn sàng (ready), nghẽn (blocked) và kết thúc (Dead). Một luồng thì được xem như là một quá trình nhẹ.



Hình 2.1 Các trạng thái của Luồng

Nhờ vào luồng, người ta thiết kế các server có thể đáp ứng nhiều yêu cầu một cách đồng thời.



Hình 2.2 - Sử dụng luồng cho các server

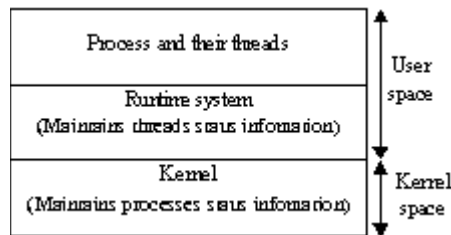
Trong mô hình này, Server có một **Luồng phân phát** (Dispatcher thread) và nhiều **Luồng thực hiện** (Worker thread). Luồng phân phát tiếp nhận các yêu cầu nối kết từ các Client, rồi chuyển chúng đến các luồng thực hiện còn rảnh để xử lý. Những luồng thực hiện hoạt động song song nhau và song song với cả luồng phân phát, nhờ đó, Server có thể phục vụ nhiều Client một cách đồng thời.

### 1.7.1. Các mức cài đặt luồng

Nhìn từ góc độ hệ điều hành, Luồng có thể được cài đặt ở một trong hai mức:

- Trong không gian người dùng (user space)
- Trong không gian nhân (kernel mode):

### 1.7.1.1. Tiếp cận luồng ở mức người dùng:

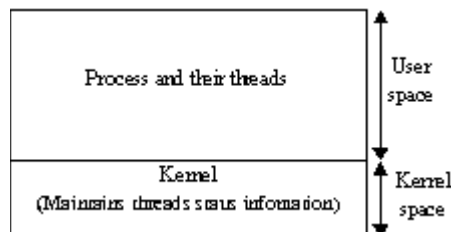


Hình 2.3 - Kiến trúc luồng cài đặt ở mức người dùng

Không gian người dùng bao gồm một hệ thống runtime mà nó tập hợp những thủ tục quản lý luồng. Các luồng chạy trong không gian nằm bên trên hệ thống runtime thì được quản lý bởi hệ thống này. Hệ thống runtime cũng lưu giữ một bảng tin trạng thái để theo dõi trạng thái hiện hành của mỗi luồng. Tương ứng với mỗi luồng sẽ có một mục từ trong bảng, bao gồm các thông tin về trạng thái, giá trị thanh ghi, độ ưu tiên và các thông tin khác về luồng.

Tiếp cận này có hai mức định thời biểu (Scheduling): bộ định thời biểu cho các quá trình nặng và bộ định thời biểu trong hệ thống runtime. Bộ lập biểu của hệ thống runtime chia thời gian sử dụng CPU được cấp cho một quá trình thành những khoảng nhỏ hơn để cấp cho các luồng trong quá trình đó. Như vậy việc kết thúc một luồng thì vượt ra ngoài tầm kiểm soát của kernel hệ thống.

### 1.7.1.2. Tiếp cận luồng ở mức hạt nhân hệ điều hành



Hình 2.4 - Kiến trúc luồng cài đặt ở mức hệ thống

Trong tiếp cận này không có hệ thống runtime và các luồng thì được quản lý bởi kernel của hệ điều hành. Vì vậy, bảng thông tin trạng thái của tất cả các luồng thì được lưu trữ bởi kernel. Tất cả những lời gọi mà nó làm nghẽn luồng sẽ được bẫy (TRAP) đến kernel. Khi một luồng bị nghẽn, kernel chọn luồng khác cho thực thi. Luồng được chọn có thể cùng một quá trình với luồng bị nghẽn hoặc thuộc một quá trình khác. Vì vậy sự tồn tại của một luồng thì được biết bởi kernel và chỉ có một mức lập biểu trong hệ thống.

## 1.7.2. Luồng trong java

Trong Java, luồng là 1 đối tượng thuộc lớp `java.lang.Thread`. Một chương trình trong java có thể cài đặt luồng bằng cách tạo ra một lớp con của lớp `java.lang.Thread`.

Lớp này có 3 phương thức cơ bản để điều khiển luồng là:

- `public native synchronized void start()`
- `public void run()`
- `public final void stop()`

### **Phương thức start()**

Chuẩn bị mọi thứ để thực hiện luồng.

### **Phương thức run()**

Thực hiện công việc thực sự của luồng, () sẽ được kích hoạt một cách tự động bởi phương thức start().

### **Phương thức stop()**

Kết thúc luồng.

Luồng sẽ "chết" khi tất cả các công việc trong phương thức run() được thực hiện hoặc khi phương thức stop() được kích hoạt.

### **Ví dụ**

Ví dụ sau định nghĩa lớp MyThread là một Thread có:

- Các thuộc tính
  - name: tên của thread
  - n: số lần thread xuất hiện ra màn hình
- Các phương thức:
  - MyThread(String name, int n):

Là phương thức khởi tạo, có nhiệm vụ gán giá trị cho 2 thuộc tính và gọi phương thức start() để cho thread hoạt động (start() tự động gọi run())

- run()

In n lần dòng thông báo ra màn hình rồi kết thúc thread.

- main()

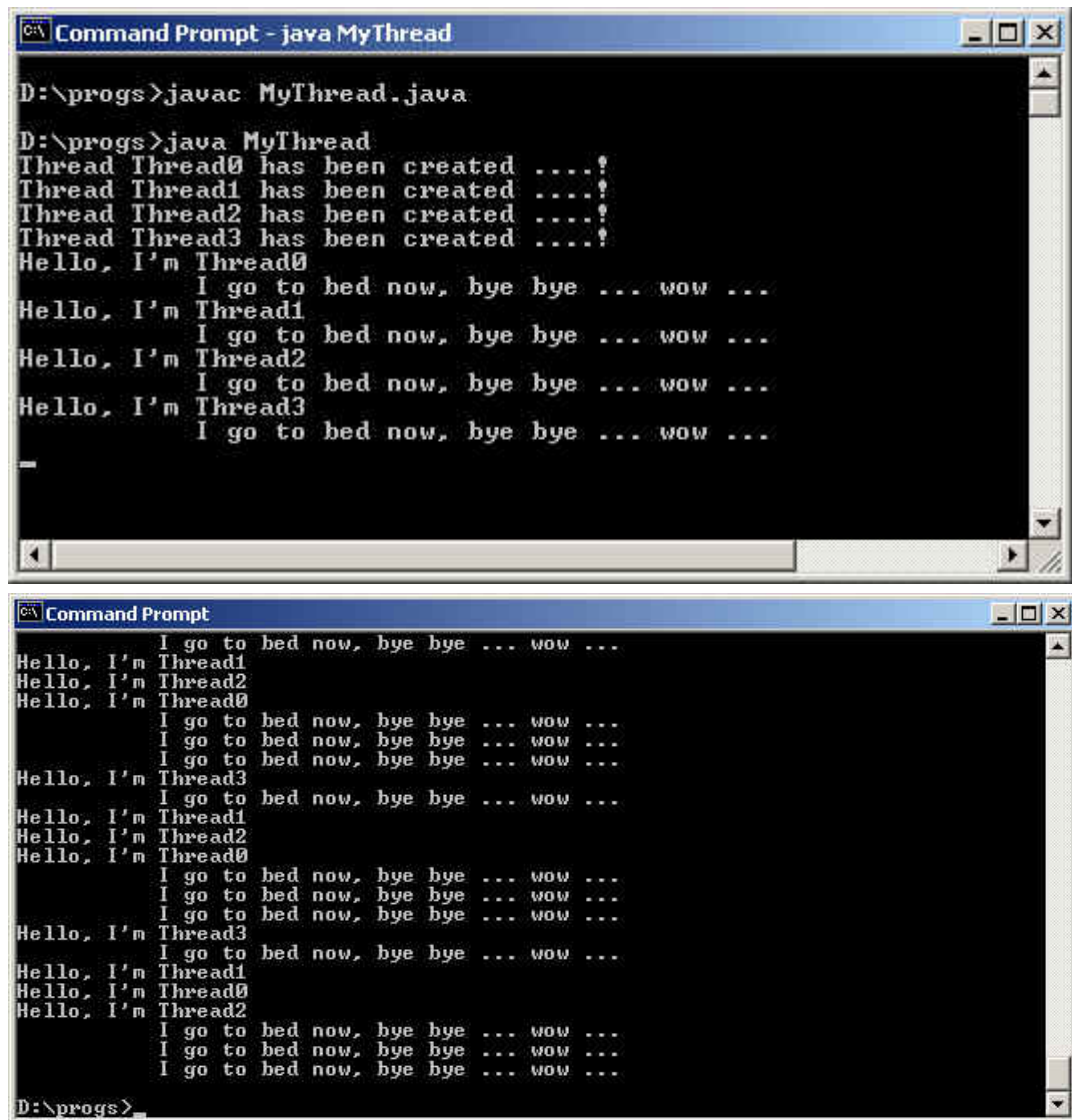
Tạo ra 4 thread thuộc lớp MyThread lần lượt có tên là Thread0, Thread1, Thread2, Thread3. Mỗi thread có 1000 lần xuất hiện ra màn hình.

Lưu chương trình sau vào tập tin MyThread.java

```
public class MyThread extends Thread{
    String name;
    int n;
    MyThread(String name, int n){
        this.name = name;
        this.n = n;
        System.out.println("Thread "+name+" has been created ....!");
        start();
    }
    public void run(){
        for(int i=0; i<n; i++) {
            System.out.println("Hello, I'm "+ name);
            System.out.println(" I go to bed now, bye bye ... wow ... ");
        }
    }
}
```

```
public static void main(String args[]){
    int n = 1000;
    int nt = 4 ;
    for (int i=0; i< nt; i++){
        MyThread t = new MyThread("Thread"+i,n);
    }
}
```

Biên dịch và thực thi ta có kết quả sau:



```
Command Prompt - java MyThread

D:\progs>javac MyThread.java

D:\progs>java MyThread
Thread Thread0 has been created ....!
Thread Thread1 has been created ....!
Thread Thread2 has been created ....!
Thread Thread3 has been created ....!
Hello, I'm Thread0
    I go to bed now, bye bye ... wow ...
Hello, I'm Thread1
    I go to bed now, bye bye ... wow ...
Hello, I'm Thread2
    I go to bed now, bye bye ... wow ...
Hello, I'm Thread3
    I go to bed now, bye bye ... wow ...
-

Command Prompt

Hello, I'm Thread1
Hello, I'm Thread2
Hello, I'm Thread0
    I go to bed now, bye bye ... wow ...
    I go to bed now, bye bye ... wow ...
    I go to bed now, bye bye ... wow ...
Hello, I'm Thread3
    I go to bed now, bye bye ... wow ...
Hello, I'm Thread1
Hello, I'm Thread2
Hello, I'm Thread0
    I go to bed now, bye bye ... wow ...
    I go to bed now, bye bye ... wow ...
    I go to bed now, bye bye ... wow ...
Hello, I'm Thread3
    I go to bed now, bye bye ... wow ...
Hello, I'm Thread1
Hello, I'm Thread0
Hello, I'm Thread2
    I go to bed now, bye bye ... wow ...
    I go to bed now, bye bye ... wow ...
    I go to bed now, bye bye ... wow ...

D:\progs>
```

Các Thread in ra màn hình theo thứ tự ta không thể xác định trước được vì chúng được thực thi song song nhau.

Để dừng tạm thời màn hình kết quả khi chương trình đang thực thi, ta nhấp chuột vào cửa sổ DOS mà chương trình đang chạy. Sau đó ta nhấn phím Enter để chương trình tiếp tục.

Ngoài ra, lớp Thread còn có 1 số các phương thức khác :

- `public static void sleep(long milliseconds)` throws `InterruptedException`: làm cho Thread bị nghẽn (Blocked) một khoảng thời gian mili giây xác định.
- `public final void suspend()`: Chuyển Thread từ trạng thái sẵn sàng sang trạng thái nghẽn.
- `public final void resume()`: Chuyển Thread từ trạng thái nghẽn sang trạng thái sẵn sàng.
- `public final void yield()` : Chuyển Thread từ trạng thái đang chạy sang trạng thái sẵn sàng.

### 1.7.2.1 Độ ưu tiên của luồng

Độ ưu tiên của các luồng xác định mức độ ưu tiên trong việc phân phối CPU giữa các luồng với nhau. Khi có nhiều luồng đang ở trạng thái "Ready", luồng có độ ưu tiên cao nhất sẽ được thực thi (chuyển đến trạng thái "running" ).

Độ ưu tiên trong Java được định nghĩa bằng các hằng số nguyên theo thứ tự giảm dần như sau:

- `Thread.MAX_PRIORITY`
- `Thread.NORM_PRIORITY`
- `Thread.MIN_PRIORITY`

Hai phương thức liên quan đến độ ưu tiên của luồng là:

- `setPriority( int x)`: Đặt độ ưu tiên của luồng là x
- `int getPriority( )`: Trả về giá trị ưu tiên của luồng

#### Ví dụ:

Trong ví dụ này chúng ta tạo ra 12 Thread thuộc lớp `MyThread`. Một mảng 3 phần tử tên **prio** chứa 3 độ ưu tiên từ cao nhất đến thấp nhất. Thread thứ *i* sẽ có độ ưu tiên ở vị trí *i*%3 trong mảng **prio**. Như vậy các thread 0,3,6,9 có độ ưu tiên cao nhất, sau đó đến Thread 1,4,7,10 và cuối cùng là các thread 2,5,8,11.

Lưu chương trình sau vào tập tin `PriorityThread.java`

```
public class PriorityThread{
    public static void main(String args[]){
        int n = 1000;
        int nt = 12;
        int prio[]={ Thread.MAX_PRIORITY,
                     Thread.NORM_PRIORITY,
                     Thread.MIN_PRIORITY};
        for (int i=0; i< nt; i++){
            MyThread t = new MyThread("Thread"+i,n);
            t.setPriority(prio[i%3]);
        }
    }
}
```







Các Thread 0,3,9 có độ ưu tiên cao hơn các Thread khác cho nên chúng được thực thi thường xuyên hơn.

Các Thread 2,5,8,11 có độ ưu tiên thấp nhất nên chúng kết thúc sau cùng.

### 1.7.3. Đồng bộ hóa giữa các luồng

Tất cả các luồng của một quá trình thì được thực thi song song và độc lập nhau nhưng lại cùng chia sẻ nhau một không gian địa chỉ của quá trình. Chính vì vậy có thể dẫn đến khả năng đụng độ trong việc cập nhật các dữ liệu dùng chung của chương trình (biến, các tập tin được mở) khi một luồng ghi lên một dữ liệu trong khi một luồng khác đang đọc dữ liệu này.

Trong trường hợp đó, cần phải sử dụng cơ chế đồng bộ hóa của Java. Có nhiều mức đồng bộ hóa như: trên một biến, trên một câu lệnh, trên một khối lệnh hay trên một phương thức.

## 1.8. Bài tập áp dụng

### Chủ đề 1: Cơ bản về Java

- **Mục đích:**
  - Sinh viên làm quen với ngôn ngữ Java, viết 1 số chương trình đơn giản bằng Java.
  - Thực tập cách nhập / xuất thông tin qua Java.
  - Thiết kế lớp đơn giản qua Java.
- **Yêu cầu**

Sinh viên thực hiện các bài tập sau

- **Bài 1** : Khảo sát cây thư mục JDK trên hệ thống máy tính đang thực tập. Đặt các biến môi trường PATH và CLASSPATH đến các vị trí thích hợp.
- **Bài 2** : Viết chương trình thể hiện ra màn hình câu : " Hello Java"
- **Bài 3** : Viết chương trình nhập vào 1 chuỗi ký tự. Đổi thành chữ Hoa và in ra màn hình.
- **Bài 4** : Viết chương trình nhập vào 1 số nguyên. Kiểm tra xem số đó có phải là số nguyên tố hay không và thông báo ra màn hình.
- **Bài 5** : Viết chương trình giải phương trình bậc 2.
- **Bài 6** : Viết chương trình tính tổng của dãy số từ 1 đến n (Với n được nhập từ bàn phím).
- **Bài 7** : Nhập vào 1 dãy số thực, tính tổng của các số dương trong dãy đó.

### Chủ đề 2: Thiết kế lớp trong Java

- **Mục đích:**
  - Thiết kế lớp dưới Java.
- **Yêu cầu**

Sinh viên thực hiện các bài tập sau

- **Bài 1** : Thiết kế lớp Diem (Điểm trong không gian 2 chiều) gồm :
  - Thành phần dữ liệu : x,y kiểu int.
  - Các hàm thành viên gồm : các phương thức khởi tạo, phương thức gán tọa độ cho 1 điểm, phương thức nhập tọa độ cho 1 điểm, phương thức in ra màn hình tọa độ điểm theo dạng (x,y), phương thức tính khoảng cách từ điểm đó đến gốc tọa độ.
  - Viết hàm main() khai thác lớp vừa định nghĩa.
- **Bài 2** : Thiết kế lớp PhanSo ( Phân số ) có:
  - 2 thuộc tính tử số và mẫu số thuộc kiểu số nguyên
  - Các phương thức: Phương thức khởi tạo, phương thức in phân số, phương thức nghịch đảo phân số, phương thức trả về giá trị thực của phân số, hàm cộng, trừ, nhân, chia 2 phân số.
  - Phương thức main() sử dụng lớp PhanSo.

### Chủ đề 3: Thread

- Mục đích:
  - Tìm hiểu về luồng (Thread), cách lập trình luồng, lập trình song song.
- Yêu cầu: Sinh viên thực bài tập sau:
  - **Bài 1** : Viết chương trình mô phỏng bài toán "Người sản xuất - Người tiêu dùng", trong đó Người sản xuất sẽ sản xuất ra một số lượng ngẫu nhiên n sản phẩm nào đó rồi yêu cầu nhập kho. Người tiêu dùng sẽ yêu cầu xuất kho một số lượng ngẫu nhiên m sản phẩm nào đó từ kho. Yêu cầu nhập kho chỉ được chấp nhận nếu số lượng hàng hóa đưa vào không vượt quá sức chứa của kho, nếu không, phải chờ cho đến khi có đủ chỗ trống trong kho. Yêu cầu xuất kho chỉ được chấp nhận khi còn đủ hàng trong kho nếu không cũng phải chờ.

Gợi ý : Thiết kế các lớp sau:

- Lớp Kho: Có thuộc tính là sức chứa, phương thức khởi tạo gán giá trị cho sức chứa, các phương thức xem số lượng hàng tồn, phương thức nhập kho, phương thức xuất kho. In thông báo mỗi khi nhập kho hay xuất kho thành công
- Lớp Người Sản Xuất là một Thread: Có thuộc tính là kho để nhập hàng. Phương thức khởi tạo gán giá trị cho kho nhập hàng. Phương thức sản xuất lặp lại công việc là tạo ra n sản phẩm ngẫu nhiên và chờ để nhập vào kho.
- Lớp Người Tiêu Dùng là một Thread: Có thuộc tính là kho để xuất hàng. Phương thức khởi tạo gán giá trị cho kho để xuất hàng. Phương thức tiêu dùng lặp lại công việc là chờ để yêu cầu xuất m sản phẩm từ kho.
- Lớp Demo tạo ra một kho và 2 người sản xuất, 2 người tiêu dùng thực hiện việc nhập xuất trên cùng một kho.

## CHƯƠNG 3

### Ống dẫn (Pipe)

#### Mục đích

Chương này nhằm giới thiệu cơ chế giao tiếp liên quá trình đầu tiên là Pipe và cách sử dụng nó trong Java để làm phương tiện giao tiếp giữa các Thread trong một chương trình.

#### Yêu cầu

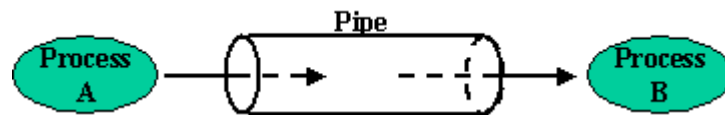
Sau khi hoàn tất chương này, bạn có thể:

- Trình bày được các đặc điểm của Pipe.
- Biết cách tạo Pipe và xuất/ nhập dữ liệu trên Pipe trong Java.
- Giải thích được chức năng của dịch vụ phản hồi thông tin (Echo Service).
- Xây dựng, biên dịch và thực thi thành công chương trình PipedEcho.

## 1.1. Giới thiệu về ống dẫn

Ống dẫn là một tiện ích được hỗ trợ trong hầu hết các ngôn ngữ lập trình vận hành trên các hệ thống đa nhiệm. Ống dẫn cho phép hai quá trình nằm trên cùng một máy có thể trao đổi dữ liệu với nhau.

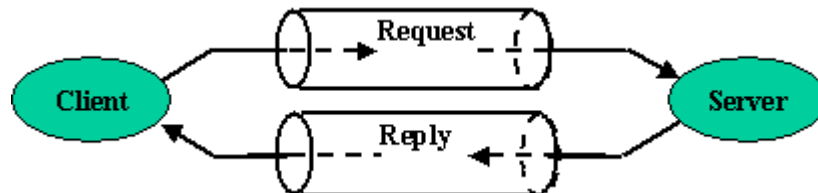
Dữ liệu đi trên ống dẫn theo một chiều nhất định. Khi sử dụng ống dẫn, người ta dùng một đầu cho việc viết dữ liệu vào và một đầu còn lại cho việc đọc dữ liệu ra.



Hình 3.1 Mô hình ống dẫn

Ống dẫn thích hợp cho trường hợp dữ liệu tạo ra của quá trình này sẽ là dữ liệu đầu vào cho quá trình kia.

Tuy nhiên ta cũng có thể sử dụng ống dẫn để xây dựng các ứng dụng theo kiến trúc Client- Server bằng cách sử dụng hai ống dẫn: một ống dẫn để truyền các yêu cầu (request), một ống dẫn để truyền các trả lời (reply).



Hình 3.2 – Dùng ống dẫn trong chương trình Client -Server

Có hai loại ống dẫn:

- Ống dẫn bình thường ( Normal Pipe): Giới hạn trong phạm vi không gian địa chỉ của một quá trình mà thôi. Nó chỉ cho phép giao tiếp giữa quá trình cha với các quá trình con hay giữa các quá trình con của một quá trình với nhau. Java hỗ trợ ống dẫn loại này. Trong đó các quá trình con được thay thế bởi các luồng.
- Ống dẫn có tên (Named Pipe): Loại này có thể cho phép hai quá trình có không gian địa chỉ khác nhau (trên cùng một máy) giao tiếp với nhau. Thực chất nó giống như một tập tin với qui định rằng dữ liệu sẽ được lấy ra ở đầu tập tin và được thêm vào ở cuối tập tin.

## 1.2. Ống dẫn trong Java

### 1.2.1. Giới thiệu

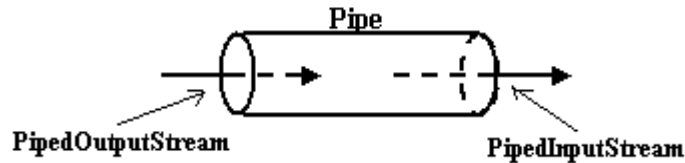
Java hỗ trợ tiện ích ống dẫn thông qua hai lớp `java.io.PipedInputStream` và `java.io.PipedOutputStream`. Chúng là hai đầu của một ống dẫn. Trong đó `PipedInputStream` là đầu đọc dữ liệu và `PipedOutputStream` là đầu ghi dữ liệu của ống dẫn.

`PipedInputStream` là lớp con của `InputStream` nên nó có tất cả các thuộc tính của `InputStream`.

PipedOutputStream là lớp con của OutputStream nên nó có tất cả các thuộc tính của OutputStream

### 1.2.2. Các cách tạo ống dẫn

Để tạo một ống dẫn ta chỉ cần tạo ra hai đối tượng thuộc lớp PipedInputStream và PipedOutputStream và nối chúng lại với nhau. Khi đó dữ liệu được ghi vào PipedOutputStream sẽ được đọc ra ở đầu PipedInputStream:



Hình 3.4 - Tạo ống dẫn trong Java

#### Cách 1

1. Tạo đầu đọc:
  - `PipedInputStream readId = new PipedInputStream();`
2. Tạo đầu ghi:
  - `PipedOutputStream writeId = new PipedOutputStream();`
3. Nối đầu đọc với đầu ghi hay ngược lại
  - `readId.connect(writeId);`
  - `// hoặc writeId.connect(readId);`

#### Cách 2

1. Tạo đầu đọc:
  - `PipedInputStream readId = new PipedInputStream();`
2. Tạo đầu ghi và nối vào đầu đọc đã có:
  - `PipedOutputStream writeId = new PipedOutputStream(readId);`

Hoặc: Ta có thể tạo đầu ghi trước rồi tạo đầu đọc sau.

**Lưu ý:** Các phương thức khởi tạo của PipedInputStream và PipedOutputStream sử dụng ở trên đòi hỏi phải "bắt" (catch) IOException do chúng có thể "quăng" ra (throws)..

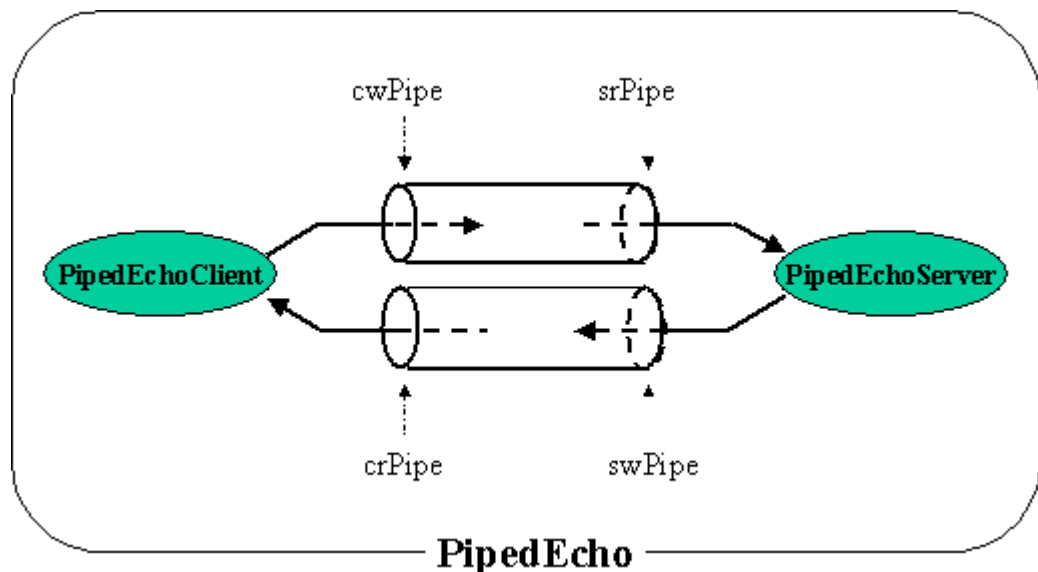
## 1.3. Dịch vụ phản hồi thông tin (Echo Service)

Dịch vụ phản hồi thông tin là một dịch vụ tồn tại trên hệ thống mạng UNIX. Dịch vụ này được xây dựng theo mô hình Client - Server, cơ chế hoạt động như sau:

- Phần Server: Chờ nhận các byte gửi đến từ Client. Với mỗi byte nhận được, Server sẽ gửi về đúng byte đã nhận trở về Client.
- Phần Client: Gửi các byte sang Server, chờ nhận các byte gửi về từ Server.

## 1.4. Giả lập dịch vụ phản hồi thông tin bằng Pipe

Phần kế tiếp ta xây dựng một ứng dụng có tên là PipeEcho mô phỏng dịch vụ phản hồi thông tin để minh họa cách sử dụng Pipe làm phương tiện giao tiếp giữa các Thread trong một ứng dụng.



Hình 3.5 – Mô hình ứng dụng PipeEcho

Trong ứng dụng này Client và Server là hai Thread thuộc lớp PipedEchoClient và PipedEchoServer. Việc trao đổi thông tin giữa client và server được thực hiện thông qua 2 Pipe (cwPipe-srPipe) và (swPipe-crPipe).

PipedEchoClient nhận các ký tự từ bàn phím, gửi chúng sang PipedEchoServer và chờ nhận các ký tự gửi về từ PipedEchoServer để in ra màn hình. PipedEchoServer chờ nhận từng ký tự từ PipedEchoClient, đổi ký tự nhận được thành ký tự hoa và gửi ngược về PipedEchoClient.

### 1.4.1. Lớp PipedEchoServer

Hãy lưu chương trình sau vào tập tin PipedEchoServer.java

```
import java.io.*;
public class PipedEchoServer extends Thread {
    PipedInputStream readPipe;
    PipedOutputStream writePipe;
    PipedEchoServer(PipedInputStream readPipe, PipedOutputStream writePipe){
        this.readPipe = readPipe;
        this.writePipe = writePipe;
        System.out.println("Server is starting . . .");
        start();
    }
    public void run(){
        while(true) {
            try{
                int ch = readPipe.read();
                ch = Character.toUpperCase((char)ch);
                writePipe.write(ch);
            }
        }
    }
}
```

```

    }
    catch (IOException ie) { System.out.println("Echo Server Error: "+ie ); }
    }
}

```

### 1.4.2. Lớp PipedEchoClient

Hãy lưu chương trình sau vào tập tin PipedEchoClient.java

```

import java.io.*;
public class PipedEchoClient extends Thread {
    PipedInputStream readPipe;
    PipedOutputStream writePipe;
    PipedEchoClient(PipedInputStream readPipe, PipedOutputStream writePipe){
        this.readPipe = readPipe;
        this.writePipe = writePipe;
        System.out.println("Client creation");
        start();
    }
    public void run(){
        while(true) {
            try {
                int ch=System.in.read();
                writePipe.write(ch);
                ch = readPipe.read();
                System.out.print((char)ch);
            }
            catch(IOException ie){
                System.out.println("Echo Client Error: "+ie );
            }
        }
    }
}

```

### 1.4.3. Lớp PipedEcho

Hãy lưu chương trình sau vào tập tin PipedEcho.java

```

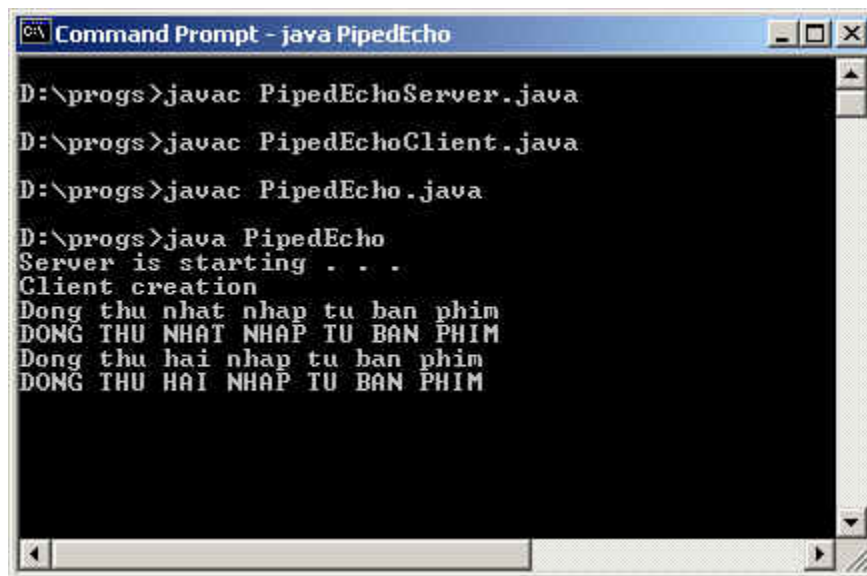
import java.io.*;
public class PipedEcho {
    public static void main(String argv[]){
        try{
            PipedOutputStream cwPipe = new PipedOutputStream();
            PipedInputStream crPipe = new PipedInputStream();
            PipedOutputStream swPipe = new PipedOutputStream(crPipe);
            PipedInputStream srPipe = new PipedInputStream(cwPipe);
            PipedEchoServer server = new PipedEchoServer(srPipe,swPipe);
            PipedEchoClient client = new PipedEchoClient(crPipe,cwPipe);
        } catch(IOException ie) {
            System.out.println("Pipe Echo Error:"+ie);
        }
    }
}

```

```
}  
}
```

#### 1.4.5. Biên dịch và thực thi chương trình

Biên dịch và thực thi chương trình theo cách sau:



```
C:\> Command Prompt - java PipedEcho  
  
D:\progs>javac PipedEchoServer.java  
D:\progs>javac PipedEchoClient.java  
D:\progs>javac PipedEcho.java  
D:\progs>java PipedEcho  
Server is starting . . .  
Client creation  
Dong thu nhap tu ban phim  
DONG THU NHAT NHAP TU BAN PHIM  
Dong thu hai nhap tu ban phim  
DONG THU HAI NHAP TU BAN PHIM
```

Nhập vào bàn phím chuỗi ký tự mà bạn muốn rồi nhấn phím Enter. Bạn sẽ thấy chuỗi ký tự in hoa của chuỗi vừa nhập xuất hiện trên màn hình.



# CHƯƠNG 4

## Socket

### Mục đích

Chương này nhằm giới thiệu về cách thức xây dựng ứng dụng Client-Server trên mạng TCP/IP theo cả hai chế độ Có nối kết (TCP) và Không nối kết (UDP).

### Yêu cầu

Sau khi hoàn tất chương này, bạn có thể:

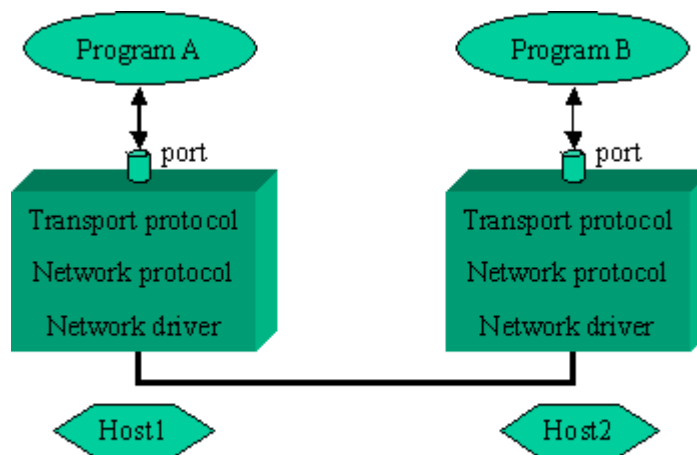
- Giải thích được Socket là gì, vai trò của số hiệu cổng (Port) và địa chỉ IP trong cơ chế Socket.
- Phân biệt được sự khác biệt của hai loại Protocol TCP và UDP.
- Trình bày được các bước xây dựng một chương trình Client-Server sử dụng Socket làm phương tiện giao tiếp trong cả hai chế độ: Có nối kết và không nối kết.
- Liệt kê các lớp hỗ trợ lập trình Socket của Java.
- Xây dựng được các chương trình Client sử dụng Sokcet ở chế độ có nối kết bằng ngôn ngữ Java.
- Xây dựng được các chương trình Server sử dụng Sokcet ở chế độ có nối kết phục vụ tuần tự và phục vụ song song bằng ngôn ngữ Java.
- Xây dựng được các chương trình Client-Server sử dụng Sokcet ở chế độ không nối kết bằng ngôn ngữ Java.
- Tự xây dựng được các Protocol mới cho ứng dụng của mình.

## 1.1. Giới thiệu về socket

### 1.1.1. Giới thiệu

Socket là một giao diện lập trình ứng dụng (API-Application Programming Interface). Nó được giới thiệu lần đầu tiên trong ấn bản UNIX - BSD 4.2. dưới dạng các hàm hệ thống theo cú pháp ngôn ngữ C (socket(), bind(), connect(), send(), receive(), read(), write(), close() ...). Ngày nay, Socket được hỗ trợ trong hầu hết các hệ điều hành như MS Windows, Linux và được sử dụng trong nhiều ngôn ngữ lập trình khác nhau: như C, C++, Java, Visual Basic, Visual C++, ...

Socket cho phép thiết lập các kênh giao tiếp mà hai đầu kênh được đánh dấu bởi hai cổng (port). Thông qua các cổng này một quá trình có thể nhận và gửi dữ liệu với các quá trình khác.



Hình 4.1 – Mô hình Socket

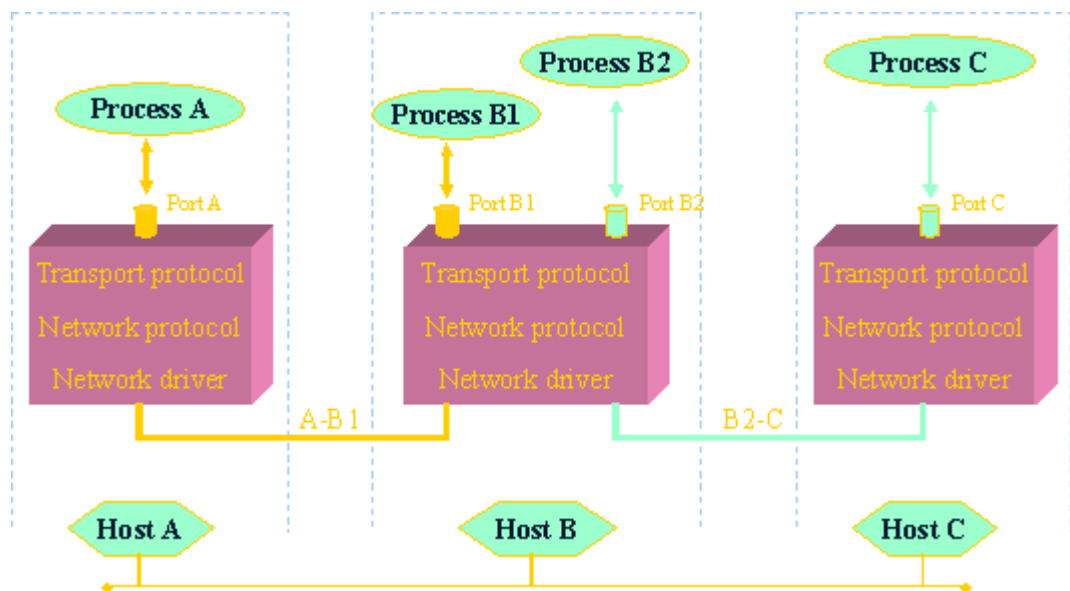
Có hai kiểu socket:

1. Socket kiểu AF\_UNIX chỉ cho phép giao tiếp giữa các quá trình trong cùng một máy tính
2. Socket kiểu AF\_INET cho phép giao tiếp giữa các quá trình trên những máy tính khác nhau trên mạng.

### 1.1.2. Số hiệu cổng (Port Number) của socket

Để có thể thực hiện các cuộc giao tiếp, một trong hai quá trình phải công bố số hiệu cổng của socket mà mình sử dụng. Mỗi cổng giao tiếp thể hiện một địa chỉ xác định trong hệ thống. Khi quá trình được gán một số hiệu cổng, nó có thể nhận dữ liệu gửi đến cổng này từ các quá trình khác. Quá trình còn lại cũng được yêu cầu tạo ra một socket.

Ngoài số hiệu cổng, hai bên giao tiếp còn phải biết địa chỉ IP của nhau. Địa chỉ IP giúp phân biệt máy tính này với máy tính kia trên mạng TCP/IP. Trong khi số hiệu cổng dùng để phân biệt các quá trình khác nhau trên cùng một máy tính.



Hình 4.2 – Cổng trong Socket

Trong hình trên, địa chỉ của quá trình B1 được xác định bằng 2 thông tin: (Host B, Port B1):

Địa chỉ máy tính có thể là địa chỉ IP dạng 203.162.36.149 hay là địa chỉ theo dạng tên miền như www.cit.ctu.edu.vn

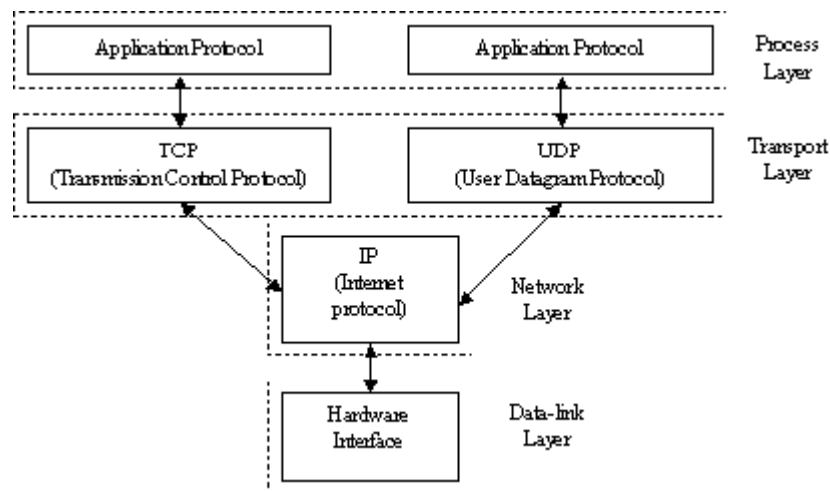
Số hiệu cổng gán cho Socket phải duy nhất trên phạm vi máy tính đó, có giá trị trong khoảng từ 0 đến 65535 (16 bits). Trong đó, các cổng từ 1 đến 1023 được gọi là cổng hệ thống được dành riêng cho các quá trình của hệ thống.

Các cổng mặc định của 1 số dịch vụ mạng thông dụng:

Số hiệu cổng	Quá trình hệ thống
7	Dịch vụ Echo
21	Dịch vụ FTP
23	Dịch vụ Telnet
25	Dịch vụ E-mail (SMTP)
80	Dịch vụ Web (HTTP)
110	Dịch vụ E-mail (POP)

### 1.1.3. Các chế độ giao tiếp

Xét kiến trúc của hệ thống mạng TCP/IP



Hình 4.3 – Bộ giao thức TCP/IP

Tầng vận chuyển giúp chuyển tiếp các thông điệp giữa các chương trình ứng dụng với nhau. Nó có thể hoạt động theo hai chế độ:

- Giao tiếp có nối kết, nếu sử dụng giao thức TCP
- Hoặc giao tiếp không nối kết, nếu sử dụng giao thức UDP

Socket là giao diện giữa chương trình ứng dụng với tầng vận chuyển. Nó cho phép ta chọn giao thức sử dụng ở tầng vận chuyển là TCP hay UDP cho chương trình ứng dụng của mình.

Bảng sau so sánh sự khác biệt giữa hai chế độ giao tiếp có nối kết và không nối kết:

Chế độ có nối kết (TCP)	Chế độ không nối kết (UDP)
<ul style="list-style-type: none"> <li>• Tồn tại kênh giao tiếp ảo giữa hai bên giao tiếp</li> <li>• Dữ liệu được gửi đi theo chế độ bảo đảm: có kiểm tra lỗi, truyền lại gói tin lỗi hay mất, bảo đảm thứ tự đến của các gói tin . . .</li> <li>• Dữ liệu chính xác, Tốc độ truyền chậm.</li> </ul>	<ul style="list-style-type: none"> <li>• Không tồn tại kênh giao tiếp ảo giữa hai bên giao tiếp</li> <li>• Dữ liệu được gửi đi theo chế độ không bảo đảm: Không kiểm tra lỗi, không phát hiện không truyền lại gói tin bị lỗi hay mất, không bảo đảm thứ tự đến của các gói tin . . .</li> <li>• Dữ liệu không chính xác, tốc độ truyền nhanh.</li> <li>• Thích hợp cho các ứng dụng cần tốc độ, không cần chính xác cao: truyền âm thanh, hình ảnh . . .</li> </ul>

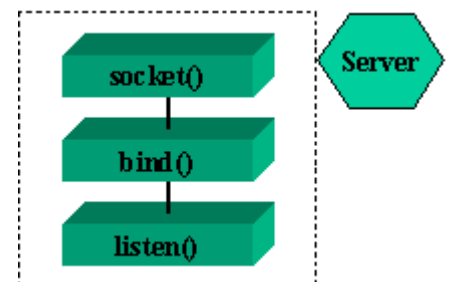
## 1.2. Xây dựng ứng dụng Client-Server với Socket

Socket là phương tiện hiệu quả để xây dựng các ứng dụng theo kiến trúc Client-Server. Các ứng dụng trên mạng Internet như Web, Email, FTP là các ví dụ điển hình.

Phần này trình bày các bước cơ bản trong việc xây dựng các ứng dụng Client-Server sử dụng Socket làm phương tiện giao tiếp theo cả hai chế độ: Có nối kết và không nối kết.

### 1.2.1. Mô hình Client-Server sử dụng Socket ở chế độ có nối kết (TCP)

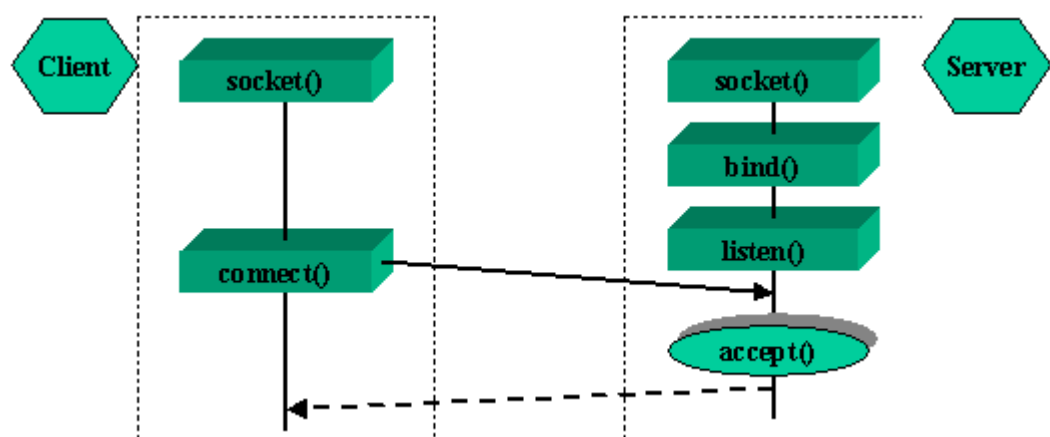
**Giai đoạn 1:** Server tạo Socket, gán số hiệu cổng và lắng nghe yêu cầu nối kết.



- `socket()`: Server yêu cầu tạo một socket để có thể sử dụng các dịch vụ của tầng vận chuyển.
- `bind()`: Server yêu cầu gán số hiệu cổng (port) cho socket.
- `listen()`: Server lắng nghe các yêu cầu nối kết từ các client trên cổng đã được gán.

Server sẵn sàng phục vụ Client.

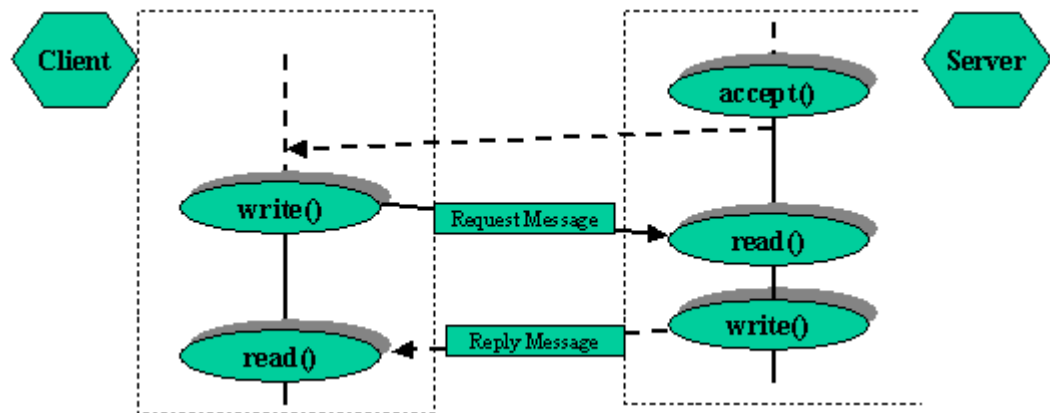
**Giai đoạn 2:** Client tạo Socket, yêu cầu thiết lập một nối kết với Server.



- `socket()`: Client yêu cầu tạo một socket để có thể sử dụng các dịch vụ của tầng vận chuyển, thông thường hệ thống tự động gán một số hiệu cổng còn rảnh cho socket của Client.
- `connect()`: Client gửi yêu cầu nối kết đến server có địa chỉ IP và Port xác định.

- `accept()`: Server chấp nhận nối kết của client, khi đó một kênh giao tiếp ảo được hình thành, Client và server có thể trao đổi thông tin với nhau thông qua kênh ảo này.

### Giai đoạn 3: Trao đổi thông tin giữa Client và Server.

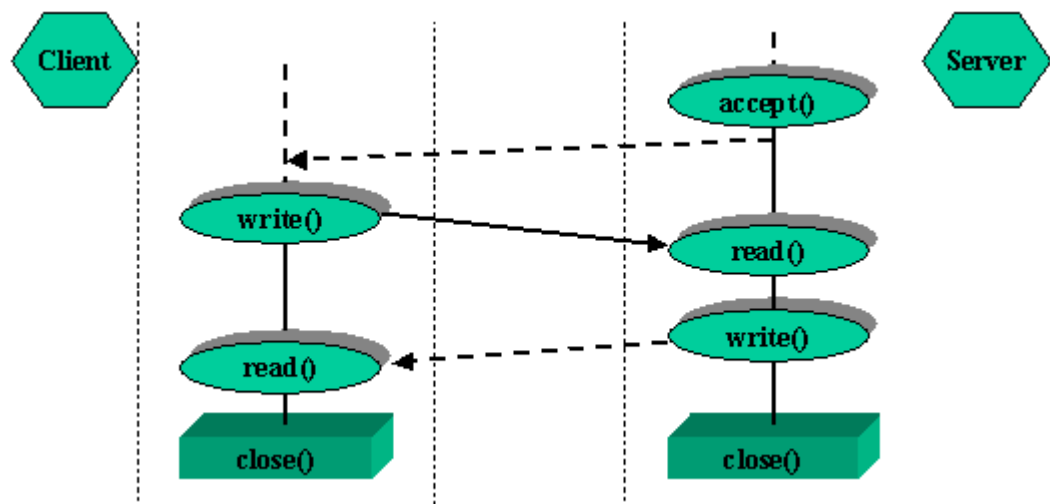


- Sau khi chấp nhận yêu cầu nối kết, thông thường server thực hiện lệnh `read()` và chờ cho đến khi có thông điệp yêu cầu (Request Message) từ client gửi đến.
- Server phân tích và thực thi yêu cầu. Kết quả sẽ được gửi về client bằng lệnh `write()`.
- Sau khi gửi yêu cầu bằng lệnh `write()`, client chờ nhận thông điệp kết quả (ReplyMessage) từ server bằng lệnh `read()`.

Trong giai đoạn này, việc trao đổi thông tin giữa Client và Server phải tuân thủ giao thức của ứng dụng (Dạng thức và ý nghĩa của các thông điệp, qui tắc bắt tay, đồng bộ hóa,...). Thông thường Client sẽ là người gửi yêu cầu đến Server trước.

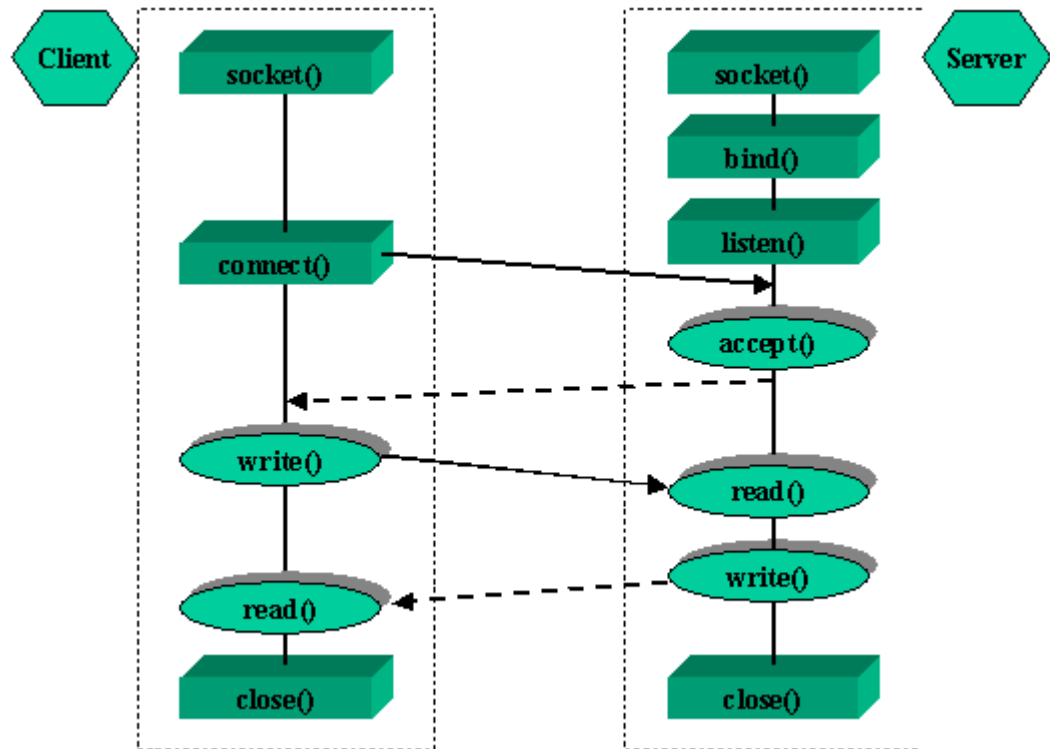
Nếu chúng ta phát triển ứng dụng theo các Protocol đã định nghĩa sẵn, chúng ta phải tham khảo và tuân thủ đúng những qui định của giao thức. Bạn có thể tìm đọc mô tả chi tiết của các Protocol đã được chuẩn hóa trong các tài liệu RFC (Request For Comments). Ngược lại, nếu chúng ta phát triển một ứng dụng Client-Server riêng của mình, thì công việc đầu tiên chúng ta phải thực hiện là đi xây dựng Protocol cho ứng dụng.

### Giai đoạn 4: Kết thúc phiên làm việc.



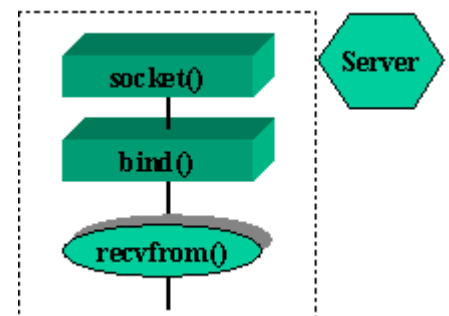
- Các câu lệnh read(), write() có thể được thực hiện nhiều lần (ký hiệu bằng hình ellipse).
- Kênh ảo sẽ bị xóa khi Server hoặc Client đóng socket bằng lệnh close().

Như vậy toàn bộ tiến trình diễn ra như sau:



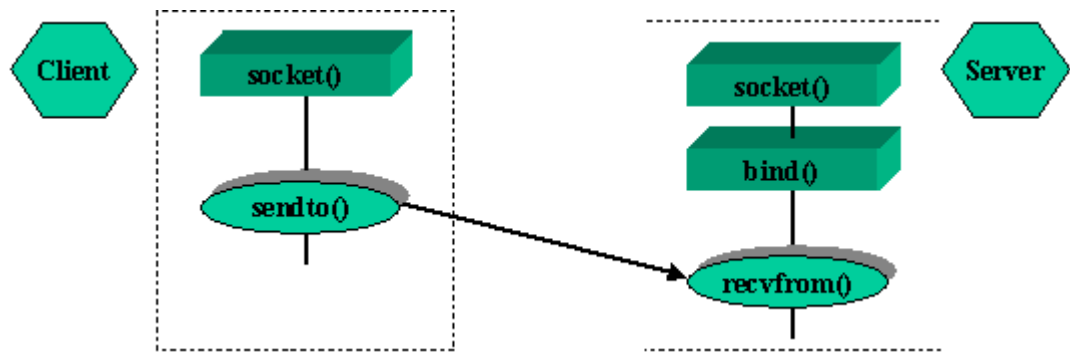
### 1.2.2. Mô hình Client-Server sử dụng Socket ở chế độ không nối kết (UDP)

**Giai đoạn 1:** Server tạo Socket - gán số hiệu cổng.

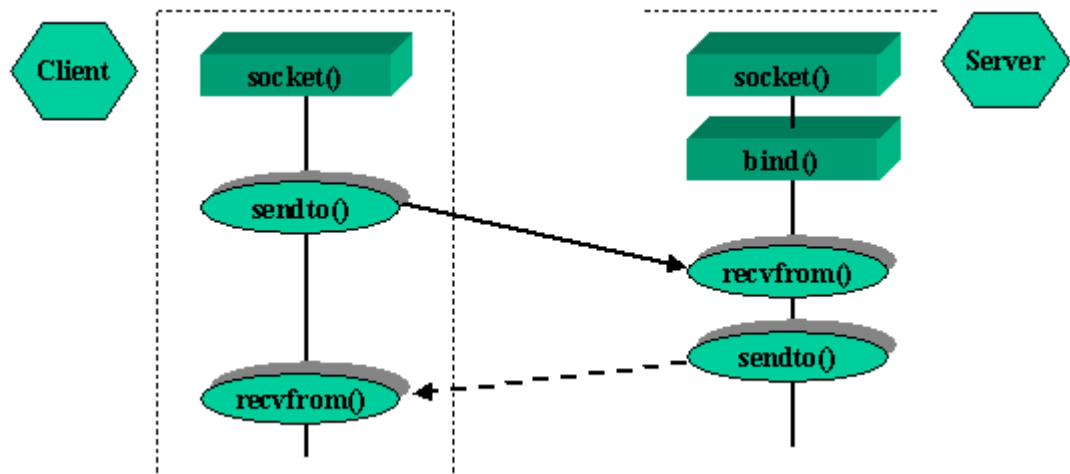


- socket(): Server yêu cầu tạo một socket để có thể sử dụng các dịch vụ của tầng vận chuyển.
- bind(): Server yêu cầu gán số hiệu cổng cho socket..

**Giai đoạn 2:** Client tạo Socket.



**Giai đoạn 3:** Trao đổi thông tin giữa Client và Server.



Sau khi tạo Socket xong, Client và Server có thể trao đổi thông tin qua lại với nhau thông qua hai hàm `sendto()` và `recvfrom()`. Đơn vị dữ liệu trao đổi giữa Client và Server là các **Datagram Package** (Gói tin thư tín). Protocol của ứng dụng phải định nghĩa khuôn dạng và ý nghĩa của các Datagram Package. Mỗi Datagram Package có chứa thông tin về địa chỉ người gửi và người nhận (IP, Port).

### 1.3. Socket dưới ngôn ngữ Java

Java hỗ trợ lập trình mạng thông qua các lớp trong gói **java.net**. Một số lớp tiêu biểu được dùng cho lập trình Client-Server sử dụng socket làm phương tiện giao tiếp như:

- **InetAddress**: Lớp này quản lý địa chỉ Internet bao gồm địa chỉ IP và tên máy tính.
- **Socket**: Hỗ trợ các phương thức liên quan đến Socket cho chương trình Client ở chế độ có nối kết.
- **ServerSocket**: Hỗ trợ các phương thức liên quan đến Socket cho chương trình Server ở chế độ có nối kết.
- **DatagramSocket**: Hỗ trợ các phương thức liên quan đến Socket ở chế độ không nối kết cho cả Client và Server.
- **DatagramPacket**: Lớp cài đặt gói tin dạng thư tín người dùng (Datagram Packet) trong giao tiếp giữa Client và Server ở chế độ không nối kết.



### 1.3.1. Xây dựng chương trình Client ở chế độ có nối kết

Các bước tổng quát:

1. Mở một socket nối kết đến server đã biết địa chỉ IP (hay tên miền) và số hiệu cổng.
2. Lấy InputStream và OutputStream gắn với Socket.
3. Tham khảo Protocol của dịch vụ để định dạng đúng dữ liệu trao đổi với Server.
4. Trao đổi dữ liệu với Server nhờ vào các InputStream và OutputStream.
5. Đóng Socket trước khi kết thúc chương trình.

#### 1.3.1.1. Lớp java.net.Socket

Lớp Socket hỗ trợ các phương thức cần thiết để xây dựng các chương trình client sử dụng socket ở chế độ có nối kết. Dưới đây là một số phương thức thường dùng để xây dựng Client:

##### **public Socket(String HostName, int PortNumber) throws IOException**

Phương thức này dùng để nối kết đến một server có tên là HostName, cổng là PortNumber. Nếu nối kết thành công, một kênh ảo sẽ được hình thành giữa Client và Server.

- HostName: Địa chỉ IP hoặc tên logic theo dạng tên miền.
- PortNumber: có giá trị từ 0 ..65535

**Ví dụ:** Mở socket và nối kết đến Web Server của khoa Công Nghệ Thông Tin, Đại Học Cần Thơ:

```
Socket s = new Socket("www.cit.ctu.edu.vn",80);  
Hoặc: Socket s = new Socket("203.162.36.149",80);
```

##### **public InputStream getInputStream()**

Phương thức này trả về InputStream nối với Socket. Chương trình Client dùng InputStream này để nhận dữ liệu từ Server gửi về.

**Ví dụ:** Lấy InputStream của Socket s:

```
InputStream is = s.getInputStream();
```

##### **public OutputStream getOutputStream()**

Phương thức này trả về OutputStream nối với Socket. Chương trình Client dùng OutputStream này để gửi dữ liệu cho Server.

**Ví dụ:** Lấy OutputStream của Socket s:

```
OutputStream os = s.getOutputStream();
```

##### **public close()**

Phương thức này sẽ đóng Socket lại, giải phóng kênh ảo, xóa nối kết giữa Client và Server.

**Ví dụ:** Đóng Socket s:

```
s.close();
```

### 1.3.1.2. Chương trình TCPEchoClient

Trên hệ thống UNIX, Dịch vụ Echo được thiết kế theo kiến trúc Client-Server sử dụng Socket làm phương tiện giao tiếp. Cổng mặc định dành cho Echo Server là 7, bao gồm cả hai chế độ có nối kết và không nối kết.

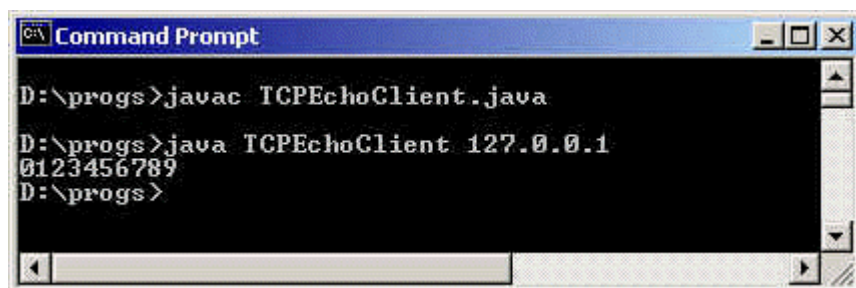
Chương trình TCPEchoClient sẽ nối kết đến EchoServer ở chế độ có nối kết, lần lượt gửi đến Echo Server 10 ký tự từ '0' đến '9', chờ nhận kết quả trả về và hiển thị chúng ra màn hình.

Hãy lưu chương trình sau vào tập tin TCPEchoClient.java

```
import java.io.*;
import java.net.Socket;

public class TCPEchoClient{
    public static void main(String args[]){
        try {
            Socket s = new Socket(args[0],7);      // Nối kết đến Server
            InputStream is = s.getInputStream();    // Lấy InputStream
            OutputStream os = s.getOutputStream(); // Lấy OutputStream
            for (int i='0'; i<='9';i++){           // Gửi '0' ->'9' đến EchoServer
                os.write(i);                       // Gửi 1 ký tự sang Server
                int ch = is.read();                 // Chờ nhận 1 ký tự từ Server
                System.out.print((char)ch);        // In ký tự nhận được ra màn hình
            }
        } //try
        catch(IOException ie){
            System.out.println("Lỗi: Không tạo được socket");
        } //catch
    } //main
}
```

Biên dịch và thực thi chương trình như sau:



Chương trình này nhận một đối số là địa chỉ IP hay tên miền của máy tính mà ở đó Echo Server đang chạy. Trong hệ thống mạng TCP/IP mỗi máy tính được gán một địa chỉ IP cục bộ là **127.0.0.1** hay có tên là **localhost**. Trong ví dụ trên, chương trình Client nối kết đến Echo Server trên cùng máy với nó.

### 1.3.2. Xây dựng chương trình Server ở chế độ có nối kết

#### 1.3.2.1. Lớp java.net.ServerSocket

Lớp ServerSocket hỗ trợ các phương thức cần thiết để xây dựng các chương trình Server sử dụng socket ở chế độ có nối kết. Dưới đây là một số phương thức thường dùng để xây dựng Server:

##### **public ServerSocket(int PortNumber);**

Phương thức này tạo một Socket với số hiệu cổng là PortNumber mà sau đó Server sẽ lắng nghe trên cổng này.

**Ví dụ:** Tạo socket cho Server với số hiệu cổng là 7:

```
ServerSocket ss = new ServerSocket(7);
```

##### **public Socket accept();**

Phương thức này lắng nghe yêu cầu nối kết của các Client. Đây là một phương thức hoạt động ở chế độ nghẽn. Nó sẽ bị nghẽn cho đến khi có một yêu cầu nối kết của client gửi đến.

Khi có yêu cầu nối kết của Client gửi đến, nó sẽ chấp nhận yêu cầu nối kết, trả về một Socket là một đầu của kênh giao tiếp ảo giữa Server và Client yêu cầu nối kết.

**Ví dụ:** Socket ss chờ nhận yêu cầu nối kết:

```
Socket s = ss.accept();
```

Server sau đó sẽ lấy InputStream và OutputStream của Socket mới s để giao tiếp với Client.

#### 1.3.2.2. Xây dựng chương trình Server phục vụ tuần tự

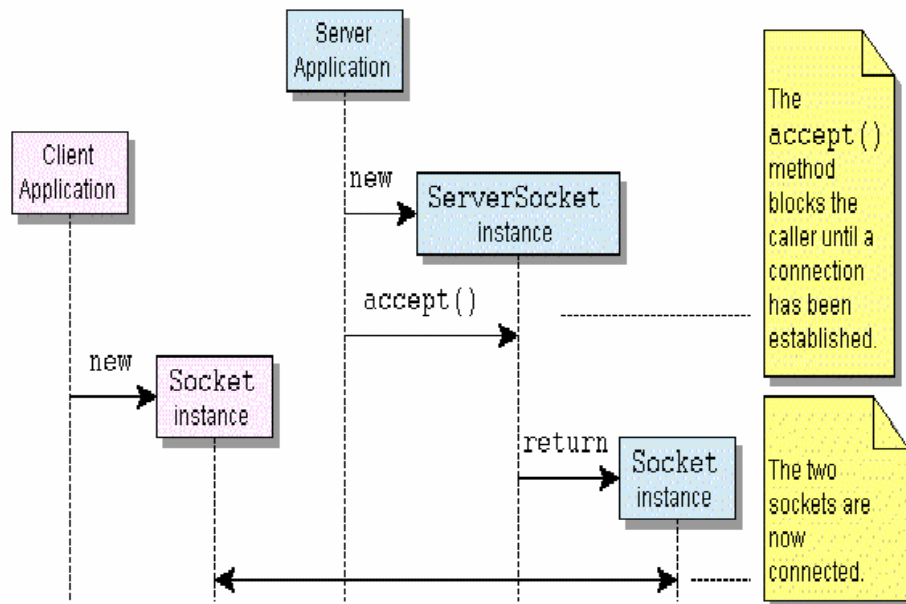
Một Server có thể được cài đặt để phục vụ các Client theo hai cách: phục vụ tuần tự hoặc phục vụ song song.

Trong chế độ phục vụ tuần tự, tại một thời điểm Server chỉ chấp nhận một yêu cầu nối kết. Các yêu cầu nối kết của các Client khác đều không được đáp ứng (đưa vào hàng đợi).

Ngược lại trong chế độ phục vụ song song, tại một thời điểm Server chấp nhận nhiều yêu cầu nối kết và phục vụ nhiều Client cùng lúc.

#### **Các bước tổng quát của một Server phục vụ tuần tự**

1. Tạo socket và gán số hiệu cổng cho server.
2. Lắng nghe yêu cầu nối kết.
3. Với một yêu cầu nối kết được chấp nhận thực hiện các bước sau:
  - Lấy InputStream và OutputStream gắn với Socket của kênh ảo vừa được hình thành.
  - Lặp lại công việc sau:
    - Chờ nhận các yêu cầu (công việc).
    - Phân tích và thực hiện yêu cầu.
    - Tạo thông điệp trả lời.
    - Gửi thông điệp trả lời về Client.
    - Nếu không còn yêu cầu hoặc Client kết thúc, đóng Socket và quay lại bước 2.



### 1.3.2.3. Chương trình STCPEchoServer

STCPEchoServer cài đặt một Echo Server phục vụ tuần tự ở chế độ có nối kết. Server lắng nghe trên cổng mặc định số 7.

Hãy lưu chương trình sau vào tập tin STCPEchoServer.java

```

import java.net.*;
import java.io.*;
public class STCPEchoServer {
    public final static int defaultPort = 7;
    public static void main(String[] args) {
        try {
            ServerSocket ss = new ServerSocket(defaultPort);
            while (true) {
                try {
                    Socket s = ss.accept();
                    OutputStream os = s.getOutputStream();
                    InputStream is = s.getInputStream();
                    int ch=0;
                    while(true) {
                        ch = is.read();
                        if(ch == -1) break;
                        os.write(ch);
                    }
                    s.close();
                }
                catch (IOException e) {
                    System.err.println(" Connection Error: "+e);
                }
            }
        }
        catch (IOException e) {
  
```

```

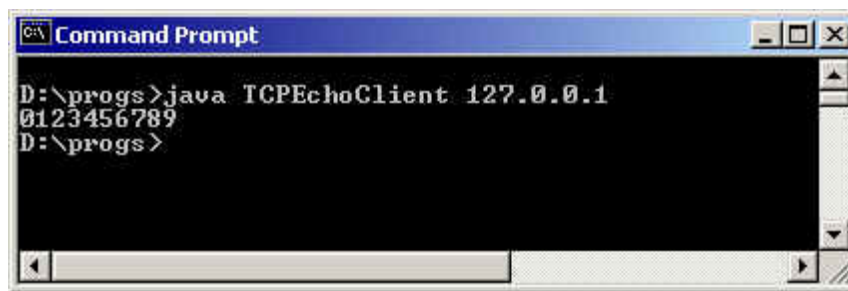
        System.err.println(" Server Creation Error:"+e);
    }
}
}

```

Biên dịch và thực thi chương trình theo cách sau:



Mở một cửa sổ DOS khác và thực thi chương trình TCPEchoClient ta có kết quả như sau:



Hai chương trình này có thể nằm trên hai máy khác nhau. Trong trường hợp đó khi thực hiện chương trình TCPEchoClient phải chú ý nhập đúng địa chỉ IP của máy tính đang chạy chương trình STCP EchoServer.

Xem địa chỉ IP của một máy tính Windows bằng lệnh **ipconfig**.

#### 1.3.2.4. Server phục vụ song song

##### Các bước tổng quát của một Server phục vụ song song

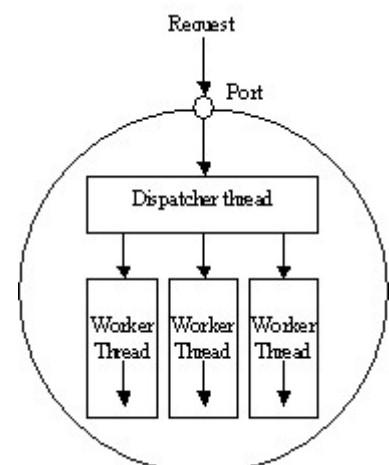
Server phục vụ song song gồm 2 phần thực hiện song song nhau:

- Phần 1: Xử lý các yêu cầu nối kết.
- Phần 2: Xử lý các thông điệp yêu cầu từ khách hàng.

Có cấu trúc như hình sau, trong đó **Phần 1** là (Dispatcher Thread), **Phần 2** là các (Worker Thread)

**Phần 1:** Lặp lại các công việc sau:

- Lắng nghe yêu cầu nối kết của khách hàng.



- Chấp nhận một yêu cầu nối kết.
  - Tạo kênh giao tiếp ảo mới với khách hàng.
  - Tạo Phần 2 để xử lý các thông điệp yêu cầu của khách hàng.

**Phần 2:** Lắp lại các công việc sau:

- Chờ nhận thông điệp yêu cầu của khách hàng.
- Phân tích và xử lý yêu cầu.
- Gửi thông điệp trả lời cho khách hàng.

Phần 2 sẽ kết thúc khi kênh ảo bị xóa đi.

Với mỗi Client, trên Server sẽ có một **Phần 2** để xử lý yêu cầu của khách hàng. Như vậy tại một thời điểm bất kỳ luôn tồn tại 1 **Phần 1** và 0 hoặc nhiều **Phần 2**.

Do Phần 2 thực thi song song với Phần 1 cho nên nó được thiết kế là một Thread.

#### 1.3.2.5. Chương trình PTCPEchoServer

PTCPEchoServer cài đặt một Echo Server phục vụ song song ở chế độ có nối kết. Server lắng nghe trên cổng mặc định là 7. Chương trình này gồm 2 lớp:

- Lớp TCPEchoServer, cài đặt các chức năng của Phần 1 - xử lý các yêu cầu nối kết của TCPEchoClient.
- Lớp RequestProcessing, là một Thread cài đặt các chức năng của Phần 2 - Xử lý các thông điệp yêu cầu.

Hãy lưu chương trình sau vào tập tin PTCPEchoServer.java

```
import java.net.*;
import java.io.*;
public class PTCPEchoServer {
    public final static int defaultPort = 7; // Cổng mặc định
    public static void main(String[] args) {
        try {
            ServerSocket ss = new ServerSocket(defaultPort); //Tạo socket cho server
            while (true) {
                try {
                    Socket s = ss.accept(); // Lắng nghe các yêu cầu nối kết
                    RequestProcessing rp = new RequestProcessing(s); // Tạo phần xử lý
                    rp.start(); // Khởi động phần xử lý cho Client hiện tại
                }
                catch (IOException e) {
                    System.out.println("Connection Error: "+e);
                }
            }
        }
        catch (IOException e) {
            System.err.println("Create Socket Error: "+e);
        }
    }
}
```

```
class RequestProcessing extends Thread {
    Socket channel; //Socket của kênh ảo nối với Client hiện tại
    public RequestProcessing(Socket s){
        channel = s; // Nhận socket của kênh ảo nối với Client
    }
    public void run() {
        try {
            OutputStream os = channel.getOutputStream();
            InputStream is = channel.getInputStream();
            while (true) {
                int n = is.read();    // Nhận ký tự từ Client
                if (n == -1) break;   // Thoát nếu kênh ảo bị xóa
                os.write(n);          // Gởi ký tự nhận được về Client
            }
        }
        catch (IOException e) {
            System.err.println("Request Processing Error: "+e);
        }
    }
}
```

Biên dịch và thực thi chương trình như sau:



Sau đó mở thêm 2 cửa sổ DOS khác để thực thi chương trình TCPEchoClient nối kết tới PTCPEchoServer. Ta sẽ nhận thấy rằng PTCPEchoServer có khả năng phục vụ đồng thời nhiều Client.

### 1.3.3. Xây dựng chương trình Client - Server ở chế độ không nối kết

Khi sử dụng socket, ta có thể chọn giao thức UDP cho lớp vận chuyển. UDP viết tắt của User Datagram Protocol, cung cấp cơ chế vận chuyển không bảo đảm và không nối kết trên mạng IP, ngược với giao thức vận chuyển tin cậy, có nối kết TCP.

Cả giao thức TCP và UDP đều phân dữ liệu ra thành các gói tin. Tuy nhiên TCP có thêm vào những tiêu đề (Header) vào trong gói tin để cho phép truyền lại những gói tin thất lạc và tập hợp các gói tin lại theo thứ tự đúng đắn. UDP không cung cấp tính năng

này, nếu một gói tin bị thất lạc hoặc bị lỗi, nó sẽ không được truyền lại, và thứ tự đến đích của các gói tin cũng không giống như thứ tự lúc nó được gửi đi.

Tuy nhiên, về tốc độ, UDP sẽ truyền nhanh gấp 3 lần TCP. Cho nên chúng thường được dùng trong các ứng dụng đòi hỏi thời gian truyền tải ngắn và không cần tính chính xác cao, ví dụ truyền âm thanh, hình ảnh . . .

Mô hình client - server sử dụng lớp `ServerSocket` và `Socket` ở trên sử dụng giao thức TCP. Nếu muốn sử dụng mô hình client - server với giao thức UDP, ta sử dụng hai lớp `java.net.DatagramSocket` và `java.net.DatagramPacket`.

`DatagramSocket` được sử dụng để truyền và nhận các `DatagramPacket`. Dữ liệu được truyền đi là một mảng những byte, chúng được gói vào trong lớp `DatagramPacket`. Chiều dài của dữ liệu tối đa có thể đưa vào `DatagramPacket` là khoảng 60.000 byte (phụ thuộc vào dạng đường truyền). Ngoài ra `DatagramPacket` còn chứa địa chỉ IP và cổng của quá trình gửi và nhận dữ liệu.

Cổng trong giao thức TCP và UDP có thể trùng nhau. Trên cùng một máy tính, bạn có thể gán cổng 20 cho socket dùng giao thức TCP và cổng 20 cho socket sử dụng giao thức UDP.

### 1.3.3.1. Lớp `DatagramPacket`

Lớp này dùng để đóng gói dữ liệu gửi đi. Dưới đây là các phương thức thường sử dụng để thao tác trên dữ liệu truyền / nhận qua `DatagramSocket`.

#### **`public DatagramPacket(byte[] b, int n)`**

- Là phương thức khởi tạo, cho phép tạo ra một `DatagramPacket` chứa **n** bytes dữ liệu đầu tiên của mảng **b**. (n phải nhỏ hơn chiều dài của mảng b)
- Phương thức trả về một đối tượng thuộc lớp `DatagramPacket`

Ví dụ: Tạo `DatagramPacket` để nhận dữ liệu:

```
byte buff[] = new byte[60000]; // Nơi chứa dữ liệu nhận được
DatagramPacket inPacket = new DatagramPacket(buff, buff.length);
```

#### **`public DatagramPacket(byte[] b, int n, InetAddress ia, int port)`**

- Phương thức này cho phép tạo một `DatagramPacket` chứa dữ liệu và cả địa chỉ của máy nhận dữ liệu.
- Phương thức trả về một đối tượng thuộc lớp `DatagramPacket`

Ví dụ: Tạo `DatagramPacket` chứa chuỗi "My second UDP Packet", với địa chỉ máy nhận là [www.cit.ctu.edu.vn](http://www.cit.ctu.edu.vn), cổng của quá trình nhận là 19:

```
try { //Địa chỉ Internet của máy nhận
    InetAddress ia = InetAddress.getByName("www.cit.ctu.edu.vn");
    int port = 19; // Cổng của socket nhận
    String s = "My second UDP Packet"; // Dữ liệu gửi đi
    byte[] b = s.getBytes(); // Đổi chuỗi thành mảng bytes
    // Tạo gói tin gửi đi
    DatagramPacket outPacket = new DatagramPacket(b, b.length, ia, port);
}
catch (UnknownHostException e) {
```



```
        System.err.println(e);
    }
```

### **Các phương thức lấy thông tin trên một DatagramPacket nhận được**

Khi nhận được một DatagramPacket từ một quá trình khác gửi đến, ta có thể lấy thông tin trên DatagramPacket này bằng các phương thức sau:

- `public synchronized() InetAddress getAddress() :` Địa chỉ máy gửi
- `public synchronized() int getPort() :` Cổng của quá trình gửi
- `public synchronized() byte[] getData() :` Dữ liệu từ gói tin
- `public synchronized() int getLength() :` Chiều dài của dữ liệu trong gói tin

### **Các phương thức đặt thông tin cho gói tin gửi**

Trước khi gửi một DatagramPacket đi, ta có thể đặt thông tin trên DatagramPacket này bằng các phương thức sau:

- `public synchronized() void setAddress(InetAddress dis) :` Đặt địa chỉ máy nhận.
- `public synchronized() void setPort(int port) :` Đặt cổng quá trình nhận
- `public synchronized() void setData(byte buffer[]) :` Đặt dữ liệu gửi
- `public synchronized() void setLength(int len) :` Đặt chiều dài dữ liệu gửi

#### **1.3.3.2. Lớp DatagramSocket**

Lớp này hỗ trợ các phương thức sau để gửi / nhận các DatagramPacket

##### **`public DatagramSocket() throws SocketException`**

- Tạo Socket kiểu không nối kết cho Client. Hệ thống tự động gán số hiệu cổng chưa sử dụng cho socket.

Ví dụ: Tạo một socket không nối kết cho Client:

```
try{
    DatagramSocket ds = new DatagramSocket();
} catch(SocketException se) {
    System.out.print("Create DatagramSocket Error: "+se);
}
```

##### **`public DatagramSocket(int port) throws SocketException`**

- Tạo Socket kiểu không nối kết cho Server với số hiệu cổng được xác định trong tham số (port).

Ví dụ: Tạo một socket không nối kết cho Server với số hiệu cổng là 7:

```
try{
    DatagramSocket dp = new DatagramSocket(7);
} catch(SocketException se) {
    System.out.print("Create DatagramSocket Error: "+se);
}
```

### **public void send(DatagramPacket dp) throws IOException**

- Dùng để gửi một DatagramPacket đi.

Ví dụ: Gửi chuỗi "My second UDP Packet", cho quá trình ở địa chỉ www.cit.ctu.edu.vn, cổng nhận là 19:

```
try {
    DatagramSocket ds = new DatagramSocket(); //Tạo Socket
    //Địa chỉ Internet của máy nhận
    InetAddress ia = InetAddress.getByName("www.cit.ctu.edu.vn");
    int port = 19; // Cổng của quá trình nhận
    String s = "My second UDP Packet"; // Dữ liệu cần gửi
    byte[] b = s.getBytes(); // Đổi sang mảng bytes
    // Tạo gói tin
    DatagramPacket outPacket = new DatagramPacket(b, b.length, ia, port);
    ds.send(outPacket); // Gửi gói tin đi
}
catch (IOException e) {
    System.err.println(e);
}
```

### **public synchronized void receive(DatagramPacket dp) throws IOException**

- Chờ nhận một DatagramPacket. Quá trình sẽ bị nghẽn cho đến khi có dữ liệu đến.

Ví dụ:

```
try {
    DatagramSocket ds = new DatagramSocket(); //Tạo Socket
    byte[] b = new byte[60000]; // Nơi chứa dữ liệu nhận được
    DatagramPacket inPacket = new DatagramPacket(b, b.length); // Tạo gói tin
    ds.receive(inPacket); // Chờ nhận gói tin
}
catch (IOException e) {
    System.err.println(e);
}
```

#### **1.3.3.3. Chương trình UDPEchoServer**

Chương trình UDPEchoServer cài đặt Echo Server ở chế độ không nối kết, cổng mặc định là 7. Chương trình chờ nhận từng gói tin, lấy dữ liệu ra khỏi gói tin nhận được và gửi ngược dữ liệu đó về Client.

Lưu chương trình sau vào tập tin UDPEchoServer.java

```
import java.net.*;
import java.io.*;
public class UDPEchoServer {
    public final static int port = 7; // Cổng mặc định của Server
    public static void main(String[] args) {
        try {
            DatagramSocket ds = new DatagramSocket(port); // Tạo Socket với cổng là 7
            byte[] buffer = new byte[6000]; // Vùng đệm chứa dữ liệu cho gói tin nhận
            while(true) { // Tạo gói tin nhận
                DatagramPacket incoming = new DatagramPacket(buffer,buffer.length);
                ds.receive(incoming); // Chờ nhận gói tin gửi đến
                // Lấy dữ liệu khỏi gói tin nhận
                String theString = new String(incoming.getData(),0,incoming.getLength());
                // Tạo gói tin gửi chứa dữ liệu vừa nhận được
                DatagramPacket outgoing = new DatagramPacket(theString.getBytes(),
                    incoming.getLength(),incoming.getAddress(), incoming.getPort());
                ds.send(outgoing);
            }
        }
        catch (IOException e) {
            System.err.println(e);
        }
    }
}
```

Biên dịch và thực thi chương trình như sau



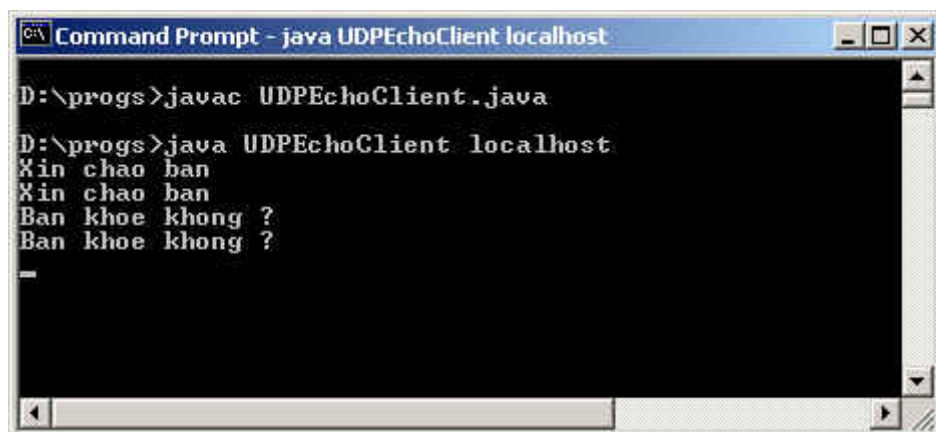
#### 1.3.3.4. Chương trình UDPEchoClient

Chương trình này cho phép người sử dụng nhận các chuỗi từ bàn phím, gửi chuỗi sang EchoServer ở chế độ không nối kết ở cổng số 7, chờ nhận và in dữ liệu từ Server gửi về ra màn hình.

Lưu chương trình sau vào tập tin UDPEchoClient.java

```
import java.net.*;
import java.io.*;
public class UDPEchoClient extends Object{
    public final static int serverPort = 7; // Cổng mặc định của Echo Server
    public static void main(String[] args) {
        try {
            if (args.length == 0) { // Kiểm tra tham số, là địa chỉ của Server
                System.out.print("Syntax: java UDPEchoClient HostName");
                return;
            }
            DatagramSocket ds = new DatagramSocket(); // Tạo DatagramSocket
            InetAddress server = InetAddress.getByName(args[0]); // Địa chỉ Server
            while(true) {
                InputStreamReader isr = new InputStreamReader(System.in); // Nhập
                BufferedReader br = new BufferedReader(isr); // một chuỗi
                String theString = br.readLine(); // từ bàn phím
                byte[] data = theString.getBytes(); // Đổi chuỗi ra mảng bytes
                // Tạo gói tin gửi
                DatagramPacket dp = new DatagramPacket(data, data.length, server, serverPort);
                ds.send(dp); // Send gói tin sang Echo Server
                byte[] buffer = new byte[6000]; // Vùng đệm cho dữ liệu nhận
                // Gói tin nhận
                DatagramPacket incoming = new DatagramPacket(buffer, buffer.length);
                ds.receive(incoming); // Chờ nhận dữ liệu từ EchoServer gửi về
                // Đổi dữ liệu nhận được dạng mảng bytes ra chuỗi và in ra màn hình
                System.out.println(new String(incoming.getData(), 0, incoming.getLength()));
            }
        }
        catch (IOException e) {
            System.err.println(e);
        }
    }
}
```

Biên dịch và thực thi chương trình như sau:



```
Command Prompt - java UDPEchoClient localhost

D:\progs>javac UDPEchoClient.java

D:\progs>java UDPEchoClient localhost
Xin chào bạn
Xin chào bạn
Bạn khỏe không ?
Bạn khỏe không ?
_
```

Chú ý, khi thực hiện chương trình UDPEchoClient phải đưa vào đối số là địa chỉ của máy tính đang thực thi chương trình UDPEchoServer. Trong ví dụ trên, Server và Client cùng chạy trên một máy nên địa chỉ của UDPEchoServer là **localhost** (hay 127.0.0.1). Nếu UDPEchoServer chạy trên máy tính khác thì khi thực thi, ta phải biết được địa chỉ IP của máy tính đó và cung cấp vào đối số của chương trình. Chẳng hạn, khi UDPEchoServer đang phục vụ trên máy tính ở địa chỉ 172.18.250.211, ta sẽ thực thi UDPEchoClient theo cú pháp sau:

```
java UDPEchoClient 172.18.250.211
```

## 1.4. Bài tập áp dụng

### Chủ đề 1: Client ở chế độ có nối kết

- **Mục đích:**

Viết các chương trình Client nối kết đến các server theo các Protocol chuẩn.

- **Yêu cầu**

Sinh viên thực hiện các bài tập sau

- **Bài 1 :** Viết chương trình nhận đối số là một URL. Nối kết đến Web Server trong URL nhận được, lấy trang web về và in ra màn hình theo dạng textfile (html).

### Chủ đề 2: Client - Server chế độ có nối kết

- **Mục đích:**

- Viết các chương trình Client -Server theo chế độ có nối kết.

- **Yêu cầu**

Sinh viên thực hiện các bài tập sau, với mỗi bài tập hãy thiết kế một Server phục vụ ở chế độ tuần tự và một Server phục vụ ở chế độ song song.

- **Bài 1:** Viết chương trình theo mô hình Client-Server sử dụng dụng Socket ở chế độ có nối kết. Trong đó :
  - + Server làm nhiệm vụ đọc một ký tự số từ '0' đến '9'.
  - ( Ví dụ : nhận số 0 : trả về "khong" , 1 : trả về "một" ; ... ... 9 : trả về "chín", nếu nhận ký tự khác số thì trả về "Không phải số nguyên" ).
  - + Client sẽ nhập vào 1 ký tự, gửi qua Server, nhận kết quả trả về từ Server và thể hiện lên màn hình
- **Bài 2:** Viết chương trình theo mô hình Client-Server sử dụng Socket ở chế độ có nối kết. Trong đó :
  - + Server sẽ nhận các yêu cầu là một chuỗi có khuôn dạng như sau:  
**"OP Operant1 Operant2\n"**
  - Trong đó:
    - OP là một ký tự chỉ phép toán muốn thực hiện: '+', '-', '\*', '/'.

- Operant1, Operant2 là đối số của phép toán.
- Các thành phần trên cách nhau bởi 1 ký tự trắng ' '.
- Kết thúc yêu cầu bằng ký tự xuống dòng '\n'.

Mỗi khi server nhận được một thông điệp nó sẽ thực hiện phép toán:  
Operant1 OP Operant2 để cho ra kết quả, sau đó đổi kết quả thành chuỗi và gửi về Client.

+ Client cho phép người dùng nhập các phép toán muốn tính theo cách thức thông thường. Ví dụ: 100+200. Client tạo ra thông điệp yêu cầu theo đúng dạng do Server qui định, mô tả về phép toán muốn Server thực thi, rồi gửi sang Server, chờ nhận kết quả trả về và in ra màn hình.

### **Chủ đề 3: Client-Server ở chế độ không nối kết**

- **Mục đích:**
  - Viết các chương trình Client -Server theo chế độ không nối kết.
- **Yêu cầu**
  - **Bài 1** : Viết chương trình Talk theo chế độ không nối kết. Cho phép hai người ngồi trên hai máy tính có thể tán gẫu (chat) với nhau.

# CHƯƠNG 5

## RPC và RMI

### Mục đích

Chương này nhằm giới thiệu cách thức xây dựng các ứng dụng phân tán bằng các cơ chế gọi thủ tục từ xa (RPC - Remote Procedure Call và RMI - Remote Method Invocation)

### Yêu cầu

Sau khi hoàn tất chương này, bạn có thể:

- Định nghĩa được ứng dụng phân tán là gì.
- Trình bày được kiến trúc của một ứng dụng phân tán xây dựng theo cơ chế gọi thủ tục từ xa (RPC).
- Trình bày được kiến trúc của một ứng dụng phân tán (hay còn gọi Ứng dụng đối tượng phân tán ) xây dựng theo cơ chế RMI của Java.
- Trình bày được các cơ chế liên quan khi xây dựng một ứng dụng theo kiểu RMI.
- Trình bày được cơ chế vận hành của một ứng dụng theo kiểu RMI.
- Giải thích được vai trò rmiregistry server.
- Liệt kê được các lớp của java hỗ trợ xây dựng các ứng dụng kiểu RMI.
- Trình bày chi tiết các bước phải qua khi xây dựng một ứng dụng theo kiểu RMI.
- Biên soạn, biên dịch và thực thi thành công chương trình minh họa Hello.
- Phân tích, thiết kế và cài đặt được các chương trình theo cơ chế RMI để giải quyết các vấn đề cụ thể.

## 1.1. Lời gọi thủ tục xa (RPC- Remote Procedure Call)

### 1.1.1. Giới thiệu

Lời gọi thủ tục xa là một cơ chế cho phép một chương trình có thể gọi thực thi một thủ tục (hay hàm) trên một máy tính khác. Trong chương trình lúc này, tồn tại hai loại thủ tục: thủ tục cục bộ và thủ tục ở xa.

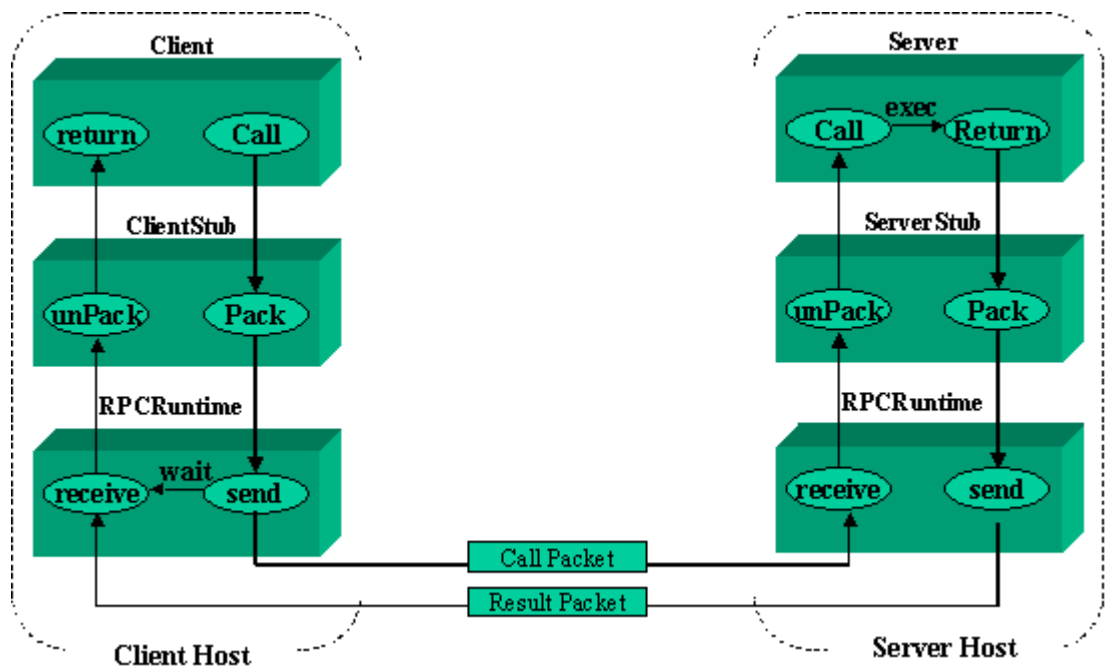
- Thủ tục cục bộ là thủ tục được định nghĩa, cài đặt và thực thi tại máy của chương trình.
- Thủ tục ở xa là thủ tục được định nghĩa, cài đặt và thực thi trên một máy tính khác.

Cú pháp giữa lời gọi thủ tục cục bộ và ở xa thì giống nhau. Tuy nhiên, khi một thủ tục ở xa được gọi đến, một thành phần của chương trình gọi là **Stub** sẽ chuyển hướng để kích hoạt một thủ tục tương ứng nằm trên một máy tính khác với máy của chương trình gọi. Đối với người lập trình, việc gọi thủ tục xa và thủ tục cục bộ thì giống nhau về mặt cú pháp. Đây chính là cơ chế cho phép đơn giản hóa việc xây dựng các ứng dụng Client-Server. Trong hệ thống RPC, Server chính là máy tính cung cấp các thủ tục ở xa cho phép các chương trình trên các máy tính khác gọi thực hiện. Client chính là các chương trình có thể gọi các thủ tục ở xa trong quá trình tính toán của mình.

Một Client có thể gọi thủ tục ở xa của nhiều hơn một máy tính. Như vậy sự thực thi của chương trình Client lúc này không còn gói gọn trên một máy tính của Client mà nó trải rộng trên nhiều máy tính khác nhau. Đây chính là mô hình của ứng dụng phân tán (Distributed Application).

### 1.1.2. Kiến trúc của chương trình Client-Server cài đặt theo cơ chế lời gọi thủ tục xa

Một ứng dụng Client-Server theo cơ chế RPC được xây dựng gồm có sáu phần như sơ đồ dưới đây:



Hình 5.1 Kiến trúc chương trình kiểu RPC



Phần Client là một quá trình người dùng, nơi khởi tạo một lời gọi thủ tục từ xa. Mỗi lời gọi thủ tục ở xa trên phần Client sẽ kích hoạt một thủ tục cục bộ tương ứng nằm trong phần Stub của Client.

Phần ClientStub cung cấp một bộ các hàm cục bộ mà phần Client có thể gọi. Mỗi một hàm của ClientStub đại diện cho một hàm ở xa được cài đặt và thực thi trên Server.

Mỗi khi một hàm nào đó của ClientStub được gọi bởi Client, ClientStub sẽ đóng gói một thông điệp để mô tả về thủ tục ở xa tương ứng mà Client muốn thực thi cùng với các tham số nếu có. Sau đó nó sẽ nhờ hệ thống RPCRuntime cục bộ gửi thông điệp này đến phần Server Stub của Server.

Phần RPCRuntime quản lý việc truyền thông điệp thông qua mạng giữa máy Client và máy Server. Nó đảm nhận việc truyền lại, báo nhận, chọn đường gói tin và mã hóa thông tin.

RPCRuntime trên máy Client nhận thông điệp yêu cầu từ ClientStub, gửi nó cho RPCRuntime trên máy Server bằng lệnh send(). Sau đó gọi lệnh wait() để chờ kết quả trả về từ Server.

Khi nhận được thông điệp từ RPCRuntime của Client gửi sang, RPCRuntime bên phía server chuyển thông điệp lên phần ServerStub.

ServerStub mở thông điệp ra xem, xác định hàm ở xa mà Client muốn thực hiện cùng với các tham số của nó. ServerStub gọi một thủ tục tương ứng nằm trên phần Server.

Khi nhận được yêu cầu của ServerStub, Server cho thực thi thủ tục được yêu cầu và gửi kết quả thực thi được cho ServerStub.

ServerStub đóng gói kết quả thực thi trong một gói tin trả lời, chuyển cho phần RPCRuntime cục bộ để nó gửi sang RPCRuntime của Client.

RPCRuntime bên phía Client chuyển gói tin trả lời nhận được cho phần ClientStub. ClientStub mở thông điệp chứa kết quả thực thi về cho Client tại vị trí phát ra lời gọi thủ tục xa.

Trong các thành phần trên, RPCRuntime được cung cấp bởi hệ thống. ClientStub và ServerStub có thể tạo ra thủ công (phải lập trình) hay có thể tạo ra bằng các công cụ cung cấp bởi hệ thống.

Cơ chế RPC được hỗ trợ bởi hầu hết các hệ điều hành mạng cũng như các ngôn ngữ lập trình.

## **1.2. Kích hoạt phương thức xa (RMI- Remote Method Invocation )**

### **1.2.1. Giới thiệu**

RMI là một sự cài đặt cơ chế RPC trong ngôn ngữ lập trình hướng đối tượng Java. Hệ thống RMI cho phép một đối tượng chạy trên một máy ảo Java này có thể kích hoạt một phương thức của một đối tượng đang chạy trên một máy ảo Java khác. Đối tượng có phương thức được gọi từ xa gọi là các đối tượng ở xa (Remote Object).

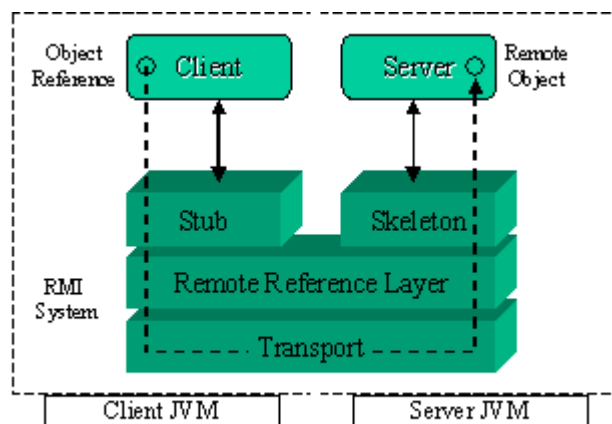
Một ứng dụng RMI thường bao gồm 2 phần phân biệt: Một chương trình Server và một chương trình Client.

- Chương trình Server tạo một số các Remote Object, tạo các **tham chiếu** (reference) đến chúng và chờ những chương trình Client kích hoạt các phương thức của các Remote Object này.
- Chương trình Client lấy một **tham chiếu** đến một hoặc nhiều Remote Object trên Server và kích hoạt các phương thức từ xa thông qua các tham chiếu.

Một chương trình Client có thể kích hoạt các phương thức ở xa trên một hay nhiều Server. Tức là sự thực thi của chương trình được trải rộng trên nhiều máy tính. Đây chính là đặc điểm của các ứng dụng phân tán. Nói cách khác, RMI là cơ chế để xây dựng các ứng dụng phân tán dưới ngôn ngữ Java.

### 1.2.2. Kiến trúc của chương trình Client-Server theo cơ chế RMI

Kiến trúc một chương trình Client-Server theo cơ chế RMI được mô tả như hình dưới đây:

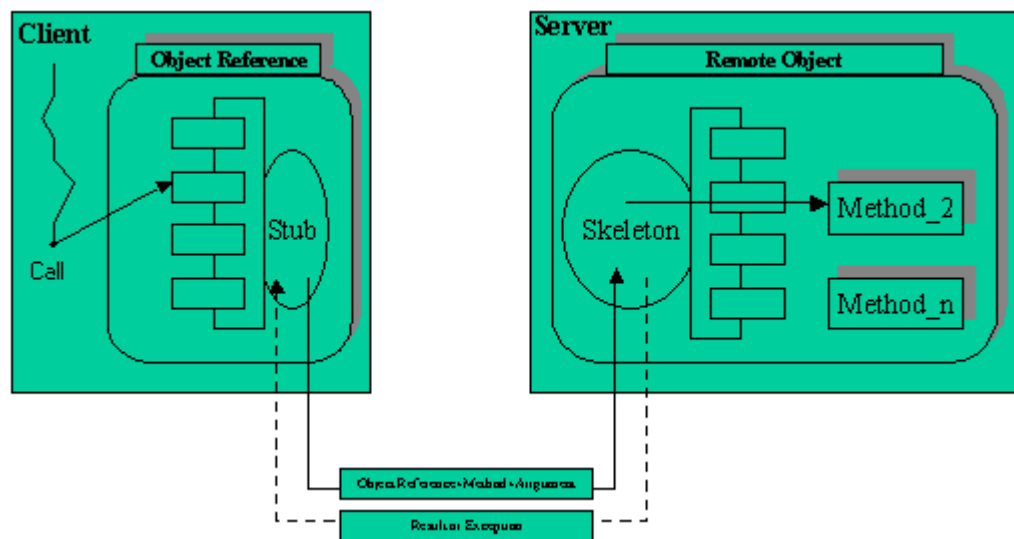


Hình 5.2 - Kiến trúc chương trình kiểu RMI

Trong đó:

- Server là chương trình cung cấp các đối tượng có thể được gọi từ xa.
- Client là chương trình có tham chiếu đến các phương thức của các đối tượng ở xa trên Server.
- Stub chứa các tham chiếu đến các phương thức ở xa trên Server.
- Skeleton đón nhận các tham chiếu từ Stub để kích hoạt phương thức tương ứng trên Server.
- Remote Reference Layer là hệ thống truyền thông của RMI.

Con đường kích hoạt một phương thức ở xa được mô tả như hình dưới đây:



Hình 5.3 Cơ chế hoạt động của RMI

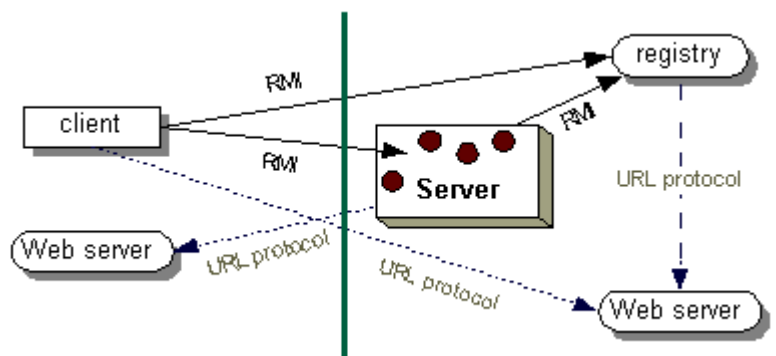
### 1.2.3. Các cơ chế liên quan trong một ứng dụng đối tượng phân tán

Trong một ứng dụng phân tán cần có các cơ chế sau:

- *Cơ chế định vị đối tượng ở xa (Locate remote objects):* Cơ chế này xác định cách thức mà chương trình Client có thể lấy được **tham chiếu** (Stub) đến các đối tượng ở xa. Thông thường người ta sử dụng một Dịch vụ danh bạ (Naming Service) lưu giữ các tham khảo đến các đối tượng cho phép gọi từ xa mà Client sau đó có thể tìm kiếm.
- *Cơ chế giao tiếp với các đối tượng ở xa (Communicate with remote objects):* Chi tiết của cơ chế giao tiếp với các đối tượng ở xa được cài đặt bởi hệ thống RMI.
- *Tải các lớp dạng bytecodes cho các lớp mà nó được chuyển tải qua lại giữa Máy ảo (Load class bytecodes for objects that are passed around):* Vì RMI cho phép các chương trình gọi phương thức từ xa trao đổi các đối tượng với các phương thức ở xa dưới dạng các tham số hay giá trị trả về của phương thức, nên RMI cần có cơ chế cần thiết để tải mã Bytecodes của các đối tượng từ máy ảo này sang máy ảo khác.

Hình dưới đây mô tả một ứng dụng phân tán dưới RMI sử dụng dịch vụ danh bạ để lấy các tham khảo của các đối tượng ở xa. Trong đó:

- Server đăng ký tên cho đối tượng có thể được gọi từ xa của mình với Dịch vụ danh bạ (Registry Server).
- Client tìm đối tượng ở xa thông qua tên đã được đăng ký trên Registry Server (looks up) và tiếp đó gọi các phương thức ở xa.



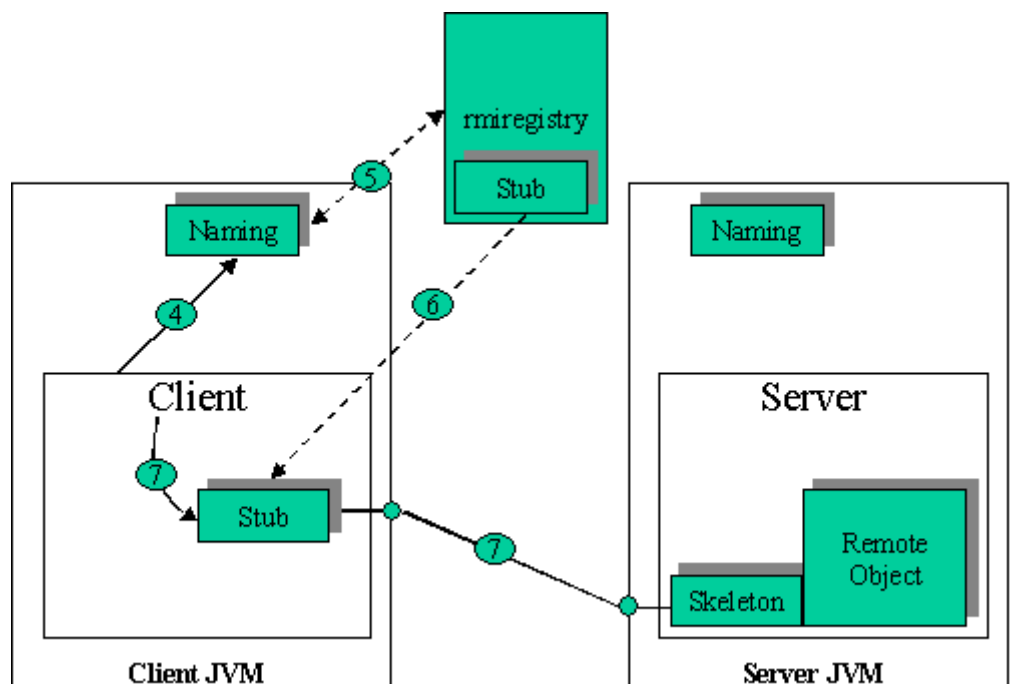
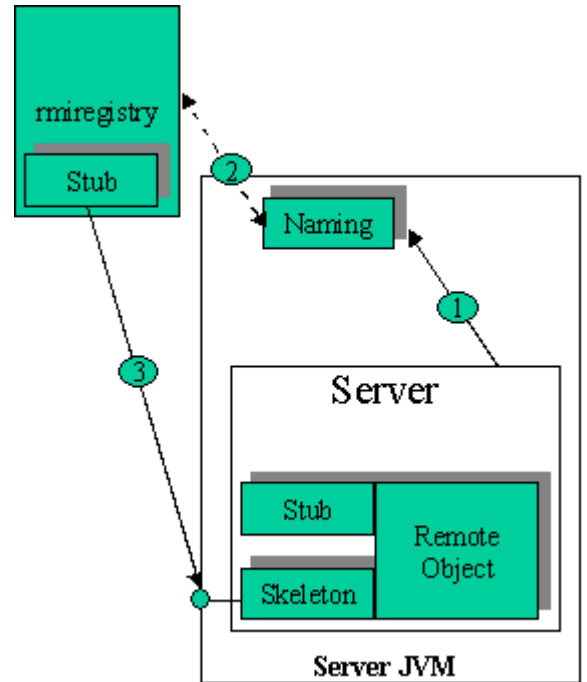
Hình 5.4 Vai trò của dịch vụ tên

- Hình minh họa cũng cho thấy cách thức mà hệ thống RMI sử dụng một WebServer sẵn có để truyền tải mã bytecodes của các lớp qua lại giữa Client và Server.

#### 1.2.4. Cơ chế vận hành của của một ứng dụng Client-Server theo kiểu RMI

Tiến trình vận hành của một ứng dụng Client-Server theo kiểu RMI diễn ra như sau:

- Bước 1: Server tạo ra các đối tượng cho phép gọi từ xa cùng với các Stub và Skeleton của chúng.
- Bước 2: Server sử dụng lớp Naming để đăng ký tên cho một đối tượng từ xa (1).
- Bước 3: Naming đăng ký Stub của đối tượng từ xa với Registry Server (2).
- Bước 4: Registry Server sẵn sàng cung cấp tham thảo đến đối tượng từ xa khi có yêu cầu (3).
- Client yêu cầu Naming định vị đối tượng xa qua tên đã được đăng ký (phương thức lookup) với dịch vụ tên (4).
- Naming tải Stub của đối tượng xa từ dịch vụ tên mà đối tượng xa đã đăng ký về Client (5).
- Cài đặt đối tượng Stub và trả về tham khảo đối tượng xa cho Client (6).
- Client thực thi một lời gọi phương thức xa thông qua đối tượng Stub (7).



### 1.2.5. Các lớp hỗ trợ chương trình theo kiểu Client-Server trong Java

Java hỗ trợ các lớp cần thiết để cài đặt các ứng dụng Client-Server theo kiểu RMI trong các gói: java.rmi. Trong số đó các lớp thường được dùng là:

- java.rmi.Naming
- java.rmi.RMISecurityManager
- java.rmi.RemoteException;
- java.rmi.server.RemoteObject
- java.rmi.server.RemoteServer
- java.rmi.server.UnicastRemoteObject

## 1.3. Xây dựng một ứng dụng phân tán với RMI

Xây dựng một ứng dụng phân tán bằng cơ chế RMI gồm các bước sau:

1. Thiết kế và cài đặt các thành phần của ứng dụng.
2. Biên dịch các chương trình nguồn và tạo ra Stub và Skeleton.
3. Tạo các lớp có thể truy xuất từ mạng cần thiết.
4. Khởi tạo ứng dụng

### 1.3.1. Thiết kế và cài đặt các thành phần của ứng dụng.

Đầu tiên bạn phải xác định lớp nào là lớp cục bộ, lớp nào là lớp được gọi từ xa. Nó bao gồm các bước sau:

- *Định nghĩa các giao diện cho các phương thức ở xa (remote interfaces):* Một remote interface mô tả các phương thức mà nó có thể được kích hoạt từ xa bởi các Client. Đi cùng với việc định nghĩa Remote Interface là việc xác định các lớp cục bộ làm tham số hay giá trị trả về của các phương thức được gọi từ xa.
- *Cài đặt các đối tượng từ xa (remote objects):* Các Remote Object phải cài đặt cho một hoặc nhiều Remote Interfaces đã được định nghĩa. Các lớp của Remote Object class cài đặt cho các phương thức được gọi từ xa đã được khai báo trong Remote Interface và có thể định nghĩa và cài đặt cho cả các phương thức được sử dụng cục bộ. Nếu có các lớp làm đối số hay giá trị trả về cho các phương thức được gọi từ xa thì ta cũng định nghĩa và cài đặt chúng.
- *Cài đặt các chương trình Client:* Các chương trình Client có sử dụng các Remote Object có thể được cài đặt ở bất kỳ thời điểm nào sau khi các Remote Interface đã được định nghĩa.

### 1.3.2. Biên dịch các tập tin nguồn và tạo Stubs và Skeleton

Giai đoạn này gồm 2 bước: Bước thứ nhất là dùng chương trình biên dịch javac để biên dịch các tập tin nguồn như các remote interface, các lớp cài đặt cho các remote interface, lớp server, lớp client và các lớp liên quan khác. Kế tiếp ta dùng trình biên dịch rmic để tạo ra stub và skeleton cho các đối tượng từ xa từ các lớp cài đặt cho các remote interface.

### 1.3.3. Tạo các lớp có thể truy xuất từ mạng

Tạo một tập tin chứa tất cả các file có liên quan như các remote interface stub, các lớp hỗ trợ mà chúng cần thiết phải tải về Client và làm cho tập tin này có thể truy cập đến thông qua một Web server.

### 1.3.4. Thực thi ứng dụng

Thực thi ứng dụng bao gồm việc thực thi rmiregistry server, thực thi server, và thực thi client.

#### Tóm lại các công việc phải làm là:

- Tạo giao diện (interface) khai báo các phương thức được gọi từ xa của đối tượng.
- Tạo lớp cài đặt (implement) cho giao diện đã được khai báo.
- Viết chương trình Server.
- Viết chương trình Client.
- Dịch các tập tin nguồn theo dạng RMI để tạo ra các lớp tương ứng và stub cho client, skeleton cho server.
- Khởi động dịch vụ registry.
- Thực hiện chương trình Server.
- Thực thi chương trình Client.

### 1.3.4. Ví dụ minh họa

Trong ví dụ này chúng ta định nghĩa một phương thức String sayHello() được gọi từ xa. Mỗi khi phương thức này được kích hoạt nó sẽ trả về chuỗi "Hello World" cho Client gọi nó.

Dưới đây là các bước để xây dựng ứng dụng:

**Bước 01: Tạo giao diện (interface) khai báo các phương thức được gọi từ xa của đối tượng.**

- Cú pháp tổng quát:

```
import java.rmi.Remote;  
import java.rmi.RemoteException;  
public interface InterfaceName extends Remote {  
    ReturnType remoteMethodOne() throws RemoteException;  
    ReturnType remoteMethodTwo() throws RemoteException;  
    ...  
}
```

- Định nghĩa remote interface có tên là HelloItf, có phương thức được gọi từ xa là String sayHello() như sau:

```
import java.rmi.Remote;  
import java.rmi.RemoteException;  
public interface HelloItf extends Remote {  
    String sayHello() throws RemoteException;  
}
```

Lưu chương trình này vào tập tin HelloItf.java

## **Bước 02: Tạo lớp cài đặt (implement) cho giao diện đã được khai báo:**

- Cú pháp tổng quát:

```
import java.rmi.server.UnicastRemoteObject;
import java.rmi.RemoteException;
public class RemoteClass extends UnicastRemoteObject implements
InterfaceName {
    public RemoteClass() throws RemoteException {
        super();
        ..... // Implement of Method
    }
    public ReturnType remoteMethodOne() throws RemoteException {
        ..... // Implement of Method
    }
    public ReturnType remoteMethodTwo() throws RemoteException {
        ..... // Definition of Method
    }
}
```

- Định nghĩa lớp cài đặt có tên là Hello cài đặt cho remote interface HelloItf

```
import java.rmi.server.UnicastRemoteObject;
import java.rmi.RemoteException;
public class Hello extends UnicastRemoteObject implements HelloItf {
    public Hello() throws RemoteException {
        super();
    }
    public String sayHello() {
        return "Hello World !";
    }
}
```

Lưu chương trình này vào tập tin Hello.java

## **Bước 03: Viết chương trình Server:**

- Cú pháp tổng quát:

```
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.RMISecurityManager;
public class ServerName {
    public static void main(String args[]) {
        if (System.getSecurityManager() == null) { // Cài đặt cơ chế bảo mật
            System.setSecurityManager(new RMISecurityManager());
        }
        try {
            // Tạo các đối tượng từ xa
        }
    }
}
```

```

RemoteClass remoteObject = new RemoteClass();

// Đăng ký tên cho các đối tượng từ xa
Naming.rebind("RegistryName", remoteObject);
...
}
catch (Exception e) {
    System.out.println("Error: . . ." + e);
}
}
}

```

- Tạo server có tên HelloServer chứa một đối tượng từ xa obj thuộc lớp cài đặt Hello. Đăng ký tên cho đối tượng obj là HelloObject

```

import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.RMISecurityManager;
public class HelloServer {
    public static void main(String args[]) {
        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new RMISecurityManager());
        }
        try {
            Hello obj = new Hello();
            Naming.rebind("HelloObject", obj);
            System.out.println("HelloObject is registried");
        }
        catch (Exception e) {
            System.out.println("Error: " + e);
        }
    }
}

```

Lưu chương trình này vào tập tin HelloServer.java

#### **Bước 04: Viết chương trình Client:**

- Cú pháp tổng quát:

```

import java.rmi.Naming;
import java.rmi.RemoteException;
public class Client {
    public static void main(String args[]) {
        String remoteObjectURL = "rmi://NameServer/RegistryName";
        InterfaceName object = null;
        try {
            object = (InterfaceName)Naming.lookup(remoteObjectURL);
            object.remoteMethodOne();
            ...
        }
        catch (Exception e) {

```



```
        System.out.println(" Error: "+ e);
    }
}
```

- Tạo client có tên là HelloClient, tìm đối tượng HelloObject trên rmiregistry chẳng hạn tại địa chỉ 172.18.211.160. Gọi phương thức sayHello() và in kết quả trả về ra màn hình.

```
import java.rmi.Naming;
import java.rmi.RemoteException;
public class HelloClient {
    public static void main(String args[]) {
        String helloURL = "rmi://172.18.211.160/HelloObject";
        HelloItf object = null;
        try {
            object = (HelloItf)Naming.lookup( helloURL);
            String message = object.sayHello();
            System.out.println(message);
        }
        catch (Exception e) {
            System.out.println("Client Error :"+ e);
        }
    }
}
```

Lưu chương trình vào tập tin HelloClient.java

**Bước 05: Dịch các tập tin nguồn theo dạng RMI để tạo ra các lớp tương ứng và stub cho client, skeleton cho server:**

- Cú pháp tổng quát:

```
javac InterfaceName.java RemoteClass.java Server.java Client.java
```

( Tạo ra các lớp InterfaceName.class RemoteClass.class Server.class Client.class)


```
rmic RemoteClass
```

( Tạo ra các lớp cho Skeleton và Stub: RemoteClass\_Skel.class RemoteClass\_Stub.class)

- Biên dịch các lớp trong Hello:

```
javac Hello.java HelloItf.java HelloServer.java HelloClient.java
```

```
rmic Hello.class
```



```
C:\> Command Prompt

D:\progs>javac Hello.java HelloItf.java HelloServer.java HelloClient.java

D:\progs>dir Hello*.class
Volume in drive D has no label.
Volume Serial Number is 5026-1F9A

Directory of D:\progs

02/08/2003  11:20a                370 Hello.class
02/08/2003  11:20a                904 HelloClient.class
02/08/2003  11:20a                215 HelloItf.class
02/08/2003  11:20a            1,049 HelloServer.class
02/04/2003  03:13a                426 HelloWorld.class
02/04/2003  03:13a            1,410 Hello_Skel.class
02/04/2003  03:13a            2,854 Hello_Stub.class
              7 File(s)              7,228 bytes
              0 Dir(s)  1,533,521,920 bytes free

D:\progs>
```

## Bước 06: Khởi động dịch vụ rmiregistry

- o Cú pháp tổng quát:

start rmiregistry [port]

Cổng mặc định là 1099.

- o Khởi động dịch vụ rmiregistry trên cổng mặc định như sau:

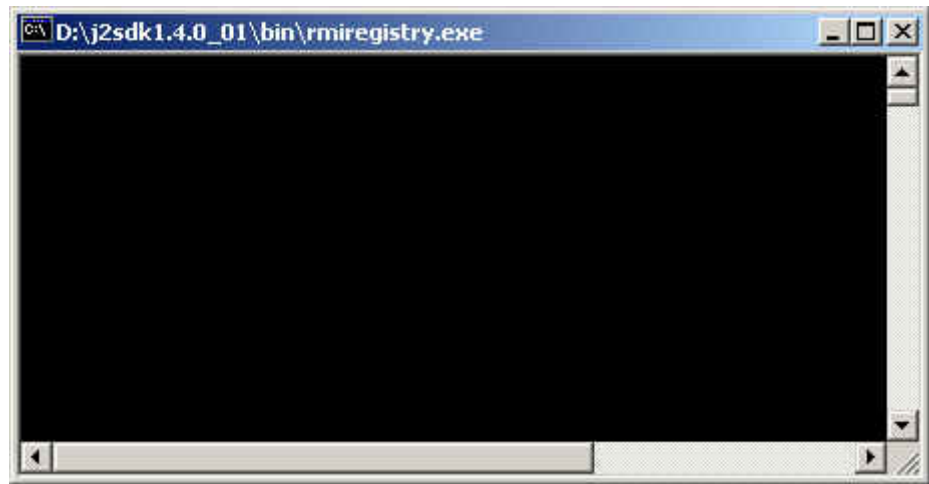


```
C:\> Command Prompt

C:\>start rmiregistry

C:\>_
```

Khi đó rmiregistry server sẽ chạy trên một cửa sổ mới, giữ nguyên cửa sổ này, không đóng nó lại.



### Bước 07: Thực hiện chương trình Server

- Cú pháp tổng quát:

```
java -Djava.security.policy=UrlOfPolicyFile ServerName
```

Trong đó UrlOfPolicyFile là địa chỉ theo dạng URL của tập tin mô tả chính sách về bảo mật mã nguồn của Server (policy file). Nó qui định "ai" (chương trình, máy tính, quá trình trên) sẽ có quyền download các tập tin của nó trong đó có stub. Để đơn giản trong phần này ta cho phép tất cả mọi người đều có quyền download các tập tin của Server. Khi triển khai các ứng dụng thật sự ta phải có các chính sách bảo mật nghiêm ngặt hơn (Tham khảo tài liệu về Security của Java). File policy có dạng như sau:

```
grant {  
    // Allow everything for now  
    permission java.security.AllPermission;  
};
```

Lưu nội dung trên vào tập tin có tên **policy.java**

- Thực thi HelloServer với địa tập tin plolicy nằm ở thư mục [D:\progs\policy.java](#)



### Bước 08: Thực thi chương trình Client:

- Cú pháp tổng quát

java ClientName

- Thực thi HelloClient với địa chỉ của rmiregistry đưa vào trong tham số

Để thực thi được chương trình HelloClient cần có hai class nằm cùng thư mục với nó là HelloItf.class và Hello\_Stub.class.



## 1.4. Bài tập áp dụng

- **Mục đích:**

Xây dựng ứng dụng phân tán theo cơ chế RMI.

- **Yêu cầu**

Sinh viên thực hiện các bài tập sau:

- **Bài 1** : Xây dựng một ứng dụng phục vụ việc bán vé máy bay cho các đại lý phân tán ở các tỉnh thành khác nhau. Ứng dụng này có các lớp sau:
  - Lớp chuyến bay: Đại diện cho một chuyến bay
    - Có các thuộc tính: Số hiệu chuyến bay, Ngày giờ bay, Nơi đi, Nơi đến, Thời gian bay, Tổng số ghế, Số lượng ghế đã bán, Số lượng ghế còn trống.
    - Các phương thức trên một chuyến bay: phương thức xem thông tin về chuyến bay, phương thức mua vé, phương thức trả vé. Để phục vụ cho nhiều đại lý các phương thức trên thuộc loại được gọi từ xa.
  - Lớp Server, tạo ra nhiều chuyến bay và duy trì nó để cho phép các đại lý thực hiện các giao dịch trên chuyến bay cụ thể.
  - Client là chương trình cho phép mỗi đại lý được quyền xem thông tin về chuyến bay, mua vé, trả vé theo yêu cầu.

## Mục lục

CHƯƠNG 1 .....	1
Tổng quan về lập trình truyền thông .....	1
1.1. Cơ chế giao tiếp liên quá trình là gì ? .....	2
1.2. Phân loại cơ chế giao tiếp liên quá trình .....	2
1.3. Mô hình tham khảo OSI .....	3
1.4. Mạng TCP/IP .....	6
1.5. Dịch vụ mạng .....	7
1.6. Mô hình Client – Server .....	7
1.6.1. Giới thiệu .....	7
1.6.2. Ví dụ về dịch vụ Web .....	8
1.6.3. Các chế độ giao tiếp .....	9
1.6.3.1. Chế độ nghẽn : .....	9
1.6.3.2. Chế độ không nghẽn: .....	9
1.7. Các kiểu kiến trúc chương trình .....	9
1.7.1. Kiến trúc đơn tầng (Single-tier Architecture) .....	10
1.7.2. Kiến trúc hai tầng (Two - Tier Architecture) .....	10
1.7.2.1. Loại Fat Client .....	11
1.7.2.2. Loại Fat Server .....	12
1.7.3. Kiến trúc đa tầng (N-Tier Architecture) .....	12
1.8. Bài tập .....	13
1.8.1. Bài tập bắt buộc .....	13
1.8.2. Bài tập gợi ý .....	13
Tìm đọc và viết một báo cáo không quá 10 trang về giao thức POP3 .....	13
CHƯƠNG 2 .....	14
Sơ lược về ngôn ngữ Java .....	14
1.1. Giới thiệu về ngôn ngữ Java .....	15
1.1.1. Lịch sử phát triển .....	15
1.1.2. Khả năng của ngôn ngữ Java .....	15
1.1.2. Những đặc điểm của ngôn ngữ Java .....	15
1.1.3. Máy ảo Java (JMV - Java Virtual Machine) .....	15
1.1.4. Hai kiểu ứng dụng dưới ngôn ngữ java .....	16
1.1.5. Bộ phát triển ứng dụng Java (JDK- Java Development Kit) .....	16
1.1.6. Kiểu dữ liệu cơ bản dưới Java .....	16
1.1.7. Các phép toán cơ bản .....	17

1.1.8. Qui cách đặt tên trong Java .....	17
1.2. Chương trình ứng dụng kiểu Application .....	18
1.2.1. Chương trình HelloWorld .....	19
1.2.3. Biên soạn chương trình bằng phần mềm Notepad của Ms Windows .....	19
1.2.4. Cài đặt bộ phát triển ứng dụng JDK.....	20
1.2.5. Biên dịch và thực thi chương trình.....	20
1.2.6. Một số ví dụ.....	21
1.2.6.1. Hiển thị thông tin ra màn hình.....	21
1.2.6.2. Đọc ký tự từ bàn phím.....	21
1.3. Các cấu trúc điều khiển trong Java.....	23
1.3.1. Lệnh if – else .....	23
1.3.2. Phép toán ? .....	24
1.3.3. Lệnh switch .....	25
1.3.4. Lệnh while.....	26
1.3.5. Lệnh do - while.....	27
1.3.6. Lệnh for .....	27
1.3.7. Lệnh break.....	28
1.3.8. Lệnh continue .....	29
1.3.9. Một số vấn đề khác.....	30
1.3.9.1. Đọc đối số của chương trình .....	30
1.3.9.2. Đổi chuỗi thành số.....	31
1.4. Ngoại lệ (EXCEPTION) .....	31
1.5. Một số vấn đề liên quan đến lớp trong Java.....	33
1.5.1. Định nghĩa lớp mới.....	33
1.5.2. Phạm vi nhìn thấy của một lớp.....	34
1.5.3. Tính thừa kế.....	35
1.6. Vào / Ra với Stream .....	36
1.6.1. Lớp java.io.InputStream.....	37
1.6.2. Lớp java.io.OutputStream .....	39
1.6.3. Nhập chuỗi từ một InputStream .....	40
1.6.4. Xuất chuỗi ra một OutputStream .....	41
1.7. Luồng (Thread).....	42
1.7.1. Các mức cài đặt luồng.....	43
1.7.1.1. Tiếp cận luồng ở mức người dùng: .....	44

1.7.1.2. Tiếp cận luồng ở mức hạt nhân hệ điều hành.....	44
1.7.2. Luồng trong java .....	44
1.7.2.1 Độ ưu tiên của luồng .....	47
1.7.3. Đồng bộ hóa giữa các luồng.....	49
1.8. Bài tập áp dụng.....	49
Chủ đề 1: Cơ bản về Java .....	49
Chủ đề 2: Thiết kế lớp trong Java .....	49
Chủ đề 3: Thread .....	50
CHƯƠNG 3.....	51
Ống dẫn (Pipe) .....	51
1.1. Giới thiệu về ống dẫn .....	52
1.2. Ống dẫn trong Java.....	52
1.2.1. Giới thiệu.....	52
1.2.2. Các cách tạo ống dẫn.....	53
1.3. Dịch vụ phản hồi thông tin (Echo Service).....	53
1.4. Giả lập dịch vụ phản hồi thông tin bằng Pipe .....	54
1.4.1. Lớp PipedEchoServer.....	54
1.4.2. Lớp PipedEchoClient .....	55
1.4.3. Lớp PipedEcho .....	55
1.4.5. Biên dịch và thực thi chương trình.....	56
CHƯƠNG 4.....	57
Socket.....	57
1.1. Giới thiệu về socket.....	58
1.1.1. Giới thiệu.....	58
1.1.2. Số hiệu cổng (Port Number) của socket.....	58
1.1.3. Các chế độ giao tiếp .....	60
1.2. Xây dựng ứng dụng Client-Server với Socket .....	61
1.2.1. Mô hình Client-Server sử dụng Socket ở chế độ có nối kết (TCP) .....	61
1.2.2. Mô hình Client-Server sử dụng Socket ở chế độ không nối kết (UDP).....	63
1.3. Socket dưới ngôn ngữ Java .....	64
1.3.1. Xây dựng chương trình Client ở chế độ có nối kết .....	65
1.3.1.1. Lớp java.net.Socket.....	65
1.3.1.2. Chương trình TCPEchoClient.....	66
1.3.2. Xây dựng chương trình Server ở chế độ có nối kết .....	67

1.3.2.1. Lớp java.net.ServerSocket .....	67
1.3.2.2. Xây dựng chương trình Server phục vụ tuần tự .....	67
1.3.2.3. Chương trình STCPEchoServer .....	68
1.3.2.4. Server phục vụ song song.....	69
1.3.2.5. Chương trình PTCPEchoServer .....	70
1.3.3. Xây dựng chương trình Client - Server ở chế độ không nối kết .....	71
1.3.3.1. Lớp DatagramPacket .....	72
1.3.3.2. Lớp DatagramSocket.....	73
1.3.3.3. Chương trình UDPEchoServer .....	74
1.3.3.4. Chương trình UDPEchoClient .....	75
1.4. Bài tập áp dụng.....	77
CHƯƠNG 5.....	79
RPC và RMI .....	79
1.1. Lời gọi thủ tục xa (RPC- Remote Procedure Call) .....	80
1.1.1. Giới thiệu.....	80
1.1.2. Kiến trúc của chương trình Client-Server cài đặt theo cơ chế lời gọi thủ tục xa	80
Hình 5.1 Kiến trúc chương trình kiểu RPC.....	80
1.2. Kích hoạt phương thức xa (RMI- Remote Method Invocation ) .....	81
1.2.1. Giới thiệu.....	81
1.2.2. Kiến trúc của chương trình Client-Server theo cơ chế RMI .....	82
1.2.3. Các cơ chế liên quan trong một ứng dụng đối tượng phân tán .....	83
1.2.4. Cơ chế vận hành của của một ứng dụng Client-Server theo kiểu RMI .....	84
1.2.5. Các lớp hỗ trợ chương trình theo kiểu Client-Server trong Java .....	85
1.3. Xây dựng một ứng dụng phân tán với RMI .....	85
1.3.1. Thiết kế và cài đặt các thành phần của ứng dụng.....	85
1.3.2. Biên dịch các tập tin nguồn và tạo Stubs và Skeleton.....	85
1.3.3. Tạo các lớp có thể truy xuất từ mạng.....	86
1.3.4. Thực thi ứng dụng .....	86
1.3.4. Ví dụ minh họa.....	86
1.4. Bài tập áp dụng.....	92
Mục lục.....	92